Twelfth Brainstorming Week on Membrane Computing

Sevilla, February 3 – 7, 2014

Luis F. Macías-Ramos
Miguel Ángel Martínez-del-Amor
Gheorghe Păun
Agustín Riscos-Núñez
Luis Valencia-Cabrera
Editors

Twelfth Brainstorming Week on Membrane Computing

Sevilla, February 3 – 7, 2014

Luis F. Macías-Ramos
Miguel Ángel Martínez-del-Amor
Gheorghe Păun
Agustín Riscos-Núñez
Luis Valencia-Cabrera
Editors

RGNC REPORT 1/2014

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2014

ISBN: 978-84-940056-4-0
Edita: Fénix Editora
c/Patricio Sáenz, 13 - 41004 Sevilla
info@fenixeditora.com
www.fenixeditora.com
Telf. (+34) 954 90 74 36
Impreso en España - Printed in Spain

Preface

The Twelfth Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from February 3 to February 7, 2014, in the organization of the Research Group on Natural Computing (RGNC) 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 all the next editions took place in Sevilla at the beginning of February, each year.

The 2014 edition of the Brainstorming had a festive character, being dedicated to the 65th birthday anniversary of Mario de Jésus Pérez-Jiménez, the enthusiastic leader of the RGNC, a member of Academia Europaea.

In the style of previous meetings in this series, the twelfth BWMC was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation. Several "provocative" talks were delivered, mainly devoted to open problems, research topics, conjectures waiting for proofs, followed by an intense cooperation among the 40 participants – see the list in the end of this preface. The efficiency of this type of meetings was again proved to be very high and the present volume illustrates this assertion.

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

A selection of papers from this volume will be considered for publication in special issues of *Fundamenta Informaticae*, which will also contain a few invited papers dedicated to Mario, on the occasion of his 65th birthday anniversary.

After each BWMC, one or two special issues of various international journals were published. Here is their list:

- BWMC 2003: *Natural Computing* – volume 2, number 3, 2003, and *New Generation Computing* – volume 22, number 4, 2004;

- BWMC 2004: *Journal of Universal Computer Science* – volume 10, number 5, 2004, and *Soft Computing* – volume 9, number 5, 2005;
- BWMC 2005: *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006);
- BWMC 2006: *Theoretical Computer Science* – volume 372, numbers 2-3, 2007;
- BWMC 2007: *International Journal of Unconventional Computing* – volume 5, number 5, 2009;
- BWMC 2008: *Fundamenta Informaticae* – volume 87, number 1, 2008;
- BWMC 2009: *International Journal of Computers, Control and Communication* – volume 4, number 3, 2009;
- BWMC 2010: *Romanian Journal of Information Science and Technology* – volume 13, number 2, 2010;
- BWMC 2011: *International Journal of Natural Computing Research* – volume 2, numbers 2-3, 2011.
- BWMC 2012: *International Journal of Computer Mathematics* – volume 99, number 4, 2013.
- BWMC 2013: *International Journal of Unconventional Computing* – volume 9, number 5-6, 2013.

Other papers elaborated during the twelfth 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. Bogdan Aman, Romanian Academy, Institute of Computer Science, Iaşi, Romania, `aman@gmail.com`
2. Fernando Arroyo, Polytechnic University of Madrid, Spain, `farroyo@eui.upm.es`
3. Juan Castellanos, Polytechnic University of Madrid, Spain, `jcastellanos@fi.upm.es`
4. Rodica Ceterchi, Faculty of Mathematics and Computer Science, University of Bucharest, Romania, `rceterchi@gmail.com`
5. Luděk Cienciala, Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic
6. Lucie Ciencialová, Research Institute of the IT4 Innovations Centre of Excellence, Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic, `lucie.ciencialova@fpf.slu.cz`
7. Erzsébet Csuhaj-Varjú, Faculty of Informatics, Eötvös Loránd University, Hungary, `csuhaj@inf.elte.hu`
8. Rudolf Freund, Technological University of Vienna, Austria, `rudifreund@gmx.at`

9. Zsolt Gazdag, Faculty of Informatics, Eötvös Loránd University, Hungary, gazdagzs@inf.elte.hu
10. Miguel A. Gutiérrez-Naranjo, University of Seville, Spain, magutier@us.es
11. Carmen Graciani, University of Seville, Spain, cgdiaz@us.es
12. Sergiu Ivanov, LACL, Université Paris Est Créteil, France, sivanov@colimite.fr
13. Alberto Leporati, University of Milan – Bicocca, Italy, leporati@disco.unimib.it
14. Jozef Kelemen, Silesian University, Opava, Czech Republic, kelemen@fpf.slu.cz
15. Alica Kelemenová, Silesian University, Opava, Czech Republic
16. Luis F. Macías-Ramos, University of Seville, Spain, lfmaciasr@us.es
17. Dusan Marcek, Silesian University, Opava, Czech Republic
18. Miguel A. Martínez-del-Amor, University of Seville, Spain, mdelamor@us.es
19. Libor Olajec, Institute of Computer Science, Silesian University, Opava, Czech Republic libor.olajec@fpf.slu.cz
20. David Orellana-Martín, University of Seville, Spain, dorelmar@gmail.com
21. Hong Peng, School of Mathematics and Computer Engineering, Xihua University City, Chengdu, China, ph.xhu@hotmail.com
22. Gheorghe Păun, Romanian Academy, Bucharest, Romania, gpaun@us.es
23. Ignacio Pérez-Hurtado, University of Seville, Spain, perezh@us.es
24. Mario de J. Pérez-Jiménez, University of Seville, Spain, marper@us.es
25. Antonio Enrico Porreca, University of Milan – Bicocca, Italy, porreca@disco.unimib.it
26. Carmen Cruz Ramos-Molinero, University of Seville, Spain, cramosmolinero@gmail.com
27. Agustín Riscos-Núñez, University of Seville, Spain, ariscosn@us.es
28. Francisco J. Romero-Campero, University of Seville, Spain, fran@us.es
29. Álvaro Romero-Jiménez, University of Seville, Spain, romero.alvaro@us.es
30. Ana Ruiz, University of Seville, Spain, anarumez@us.es
31. Jose María Sempere Luna, Polytechnical University of Valencia, Spain, jsempere@dsic.upv.es
32. Vladimír Smolka, Silesia university, Opava, Czech Republic, v.smolka@centrum.cz
33. Bosheng Song, Huazhong University of Science and Technology, Wuhan, China, boshengsong@163.com
34. Luis Valencia-Cabrera, University of Seville, Spain, lvalencia@us.es
35. Pedro Varo-Herrero, University of Seville, Spain, pevahe@gmail.com
36. György Vaszil, Faculty of Informatics, University of Debrecen, Hungary, vaszil.gyorgy@inf.unideb.hu
37. Šárka Vavrečková, Institute of Computer Science, Silesian University in Opava, Czech Republic sarka.vavreckova@fpf.slu.cz
38. Jun Wang, Electrical and Information Engineering, Xihua University, Chengdu, China, wangjun@mail.xhu.edu.cn

39. Tao Wang, School of Electrical Engineering, Southwest Jiaotong University, Chengdu, China, wangtaocdu@gmail.com
40. Claudio Zandron, University of Milan – 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) Ministerio de Economía y Competitividad (grants TIN2011-15874-E and TIN2012-37434), (ii) Instituto de Matemáticas de la Universidad de Sevilla (IMUS), (iii) Fundación para la Investigación y el Desarrollo de las Tecnologías de la Información en Andalucía (FIDETIA), (iv) V Plan Propio, Vicerrectorado de Investigación de la Universidad de Sevilla, as well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Agustín Riscos-Núñez
(April 2014)

Contents

Matter and Anti-Matter in Membrane Systems

Artiom Alhazov[1], Bogdan Aman[2], Rudolf Freund[3], Gheorghe Păun[4]

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

[2] Institute of Computer Science, Romanian Academy, Iaşi, Romania
bogdan.aman@iit.academiaromana-is.ro

[3] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
rudi@emcc.at

[4] Institute of Mathematics, Romanian Academy, Bucharest, Romania
gpaun@us.es

Summary. The concept of a matter object being annihilated when meeting its corresponding anti-matter object is investigated in the context of membrane systems, i.e., of (distributed) multiset rewriting systems applying rules in the maximally parallel way. Computational completeness can be obtained with using only non-cooperative rules besides these matter/anti-matter annihilation rules if these annihilation rules have priority over the other rules. Without this priority condition, in addition catalytic rules with one single catalyst are needed to get computational completeness. Even deterministic systems are obtained in the accepting case. Universal P systems with a rather small number of rules – 57 for computing systems, 59 for generating and 52 for accepting systems – can be constructed when using non-cooperative rules together with matter/anti-matter annihilation rules having weak priority. Allowing anti-matter objects as input and/or output, we even get a computationally complete computing model for computations on integer numbers. Interpreting sequences of symbols taken in from and/or sent out to the environment as strings, we get a model for computations on strings, which can even be interpreted as representations of elements of a group based on a computable finite presentation.

1 Introduction

Membrane systems (usually called *P systems*) can be considered as distributed multiset rewriting systems, where all objects – if possible – evolve in parallel in the membrane regions and may be communicated through the membranes. Membrane systems were introduced in [15] and since then have become an emerging field of research. Overviews can be found in the monograph [16] and the handbook of membrane systems [17]; for actual news and results we refer to the P systems

webpage [19]. Computational completeness (computing any partial recursive relation on non-negative integers) can be obtained with using cooperative rules or with catalytic rules (eventually) together with non-cooperative rules. In this paper, we use another concept to avoid cooperative rules in general: for any object a *(matter)*, we consider its anti-object *(anti-matter)* a^- as well as the corresponding *annihilation rule* $aa^- \to \lambda$, which is assumed to exist in all membranes; this annihilation rule could be assumed to remove a pair a, a^- in zero time, but here we use these annihilation rules as special non-cooperative rules having priority over all other rules in the sense of weak priority (e.g., see [1], i.e., other rules then also may be applied if objects cannot be bound by some annihilation rule any more). The idea of anti-matter has already been considered in another special variant of P systems with motivation coming from modeling neural activities, which are known as spiking neural P systems; for example, spiking neural P systems with anti-matter *(anti-spikes)* were already investigated in [14]. Moreover, in [5] the power of anti-matter for solving NP-complete problems is exhibited.

As expected (for example, compare with the Geffert normal forms, see [18]), the annihilation rules are rather powerful. Yet it is still surprising that using matter/anti-matter annihilation rules as the only non-cooperative rules, with the annihilation rules having priority, we already get computational completeness without using any catalyst; without giving the annihilation rules priority, we need one single catalyst. Even more surprising is the result that with priorities we obtain *deterministic* systems in the case of accepting P systems. Moreover, we show how rather small universal P systems with anti-matter can be obtained based on the universal register machine U_{32} constructed by Korec, see [12]. Allowing anti-matter objects as input and/or output, we even get a computationally complete computing model for computations on integer numbers. Finally, by interpreting sequences of symbols taken in from and/or sent out to the environment as strings, we also consider P systems with anti-matter as computing/accepting/generating devices for string languages or even for languages over a group based on a computable finite presentation.

2 Prerequisites

The set of integers is denoted by $\mathbb{Z}$, while the set of non-negative integers by $\mathbb{N}$. Given an alphabet V, a finite non-empty set of abstract symbols, the free monoid generated by V under the operation of concatenation is denoted by V^*. The elements of V^* are called strings, the empty string is denoted by λ, and $V^* \backslash \{\lambda\}$ is denoted by V^+. For an arbitrary alphabet $\{a_1, \ldots, a_n\}$, the number of occurrences of a symbol a_i in a string x is denoted by $|x|_{a_i}$, while the length of a string x is denoted by $|x| = \Sigma_{a_i} |x|_{a_i}$. The Parikh vector associated with x with respect to $a_1, \ldots, a_n$ is $(|x|_{a_1}, \ldots, |x|_{a_n})$. The Parikh image of an arbitrary language L over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in L, and is denoted by $Ps(L)$. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$, while for families of languages over a

one-letter (d-letter) alphabet, the corresponding sets of non-negative integers are denoted by NFL (N^dFL).

A (finite) multiset over a (finite) alphabet $V = \{a_1, \ldots, a_n\}$, is a mapping $f : V \to \mathbb{N}$ and can be represented by $\langle a_1^{f(a_1)}, \ldots, a_n^{f(a_n)} \rangle$ or by any string x for which $(|x|_{a_1}, \ldots, |x|_{a_n}) = (f(a_1), \ldots, f(a_n))$. In the following we will not distinguish between a vector $(m_1, \ldots, m_n)$, a multiset $\langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ or a string x having $(|x|_{a_1}, \ldots, |x|_{a_n}) = (m_1, \ldots, m_n)$. Fixing the sequence of symbols $a_1, \ldots, a_n$ in an alphabet V in advance, the representation of the multiset $\langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ by the string $a_1^{m_1} \ldots a_n^{m_n}$ is unique. The set of all finite multisets over an alphabet V is denoted by V°.

The family of regular and recursively enumerable string languages is denoted by REG and RE, respectively. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as [4] and [18].

Register machines.

A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and P is the set of instructions bijectively labeled by elements of B. The instructions of M can be of the following forms:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \backslash \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Increases the value of register j by one, followed by a non-deterministic jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB(j), l_2, l_3)$, with $l_1 \in B \backslash \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  If the value of register j is zero then jump to instruction l_3; otherwise, the value of register j is decreased by one, followed by a jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stops the execution of the register machine.

A *configuration* of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. Computations start by executing the instruction l_0 of P, and terminate with reaching the HALT-instruction l_h.

3 P Systems

The basic ingredients of a (cell-like) P system are the membrane structure, the multisets of objects placed in the membrane regions, and the evolution rules. The *membrane structure* is a hierarchical arrangement of membranes, in which the space between a membrane and the immediately inner membranes defines a *region/compartment*. The outermost membrane is called the *skin membrane*, the region outside is the *environment*. Each membrane can be labeled, and the label

(from a set *Lab*) will identify both the membrane and its region; the skin membrane is identified by (the label) 1. The membrane structure can be represented by an expression of correctly nested labeled parentheses, and also by a rooted tree (with the label of a membrane in each node and the skin in the root). The *multisets of objects* are placed in the compartments of the membrane structure and usually represented by strings of the form $a_1^{m_1} \dots a_n^{m_n}$.

The *evolution rules* are multiset rewriting rules of the form $u \to v$, where $u \in O^\circ$ and $v = (b_1, tar_1) \dots (b_k, tar_k)$ with $b_i \in O^\circ$ and $tar_i \in \{here, out, in\}$ or $tar_i \in \{here, out\} \cup \{in_j \mid j \in Lab\}$, $1 \leq i \leq k$. Using such a rule means "consuming" the objects of u and "producing" the objects from $b_1, \dots, b_k$ of v, where the target *here* means that the objects remain in the same region where the rule is applied, *out* means that they are sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied in the skin region), *in* means that they are sent to one of the immediately inner membranes, chosen in a non-deterministic way, and in_j means that they are sent into the specified inner membrane. In general, the target indication *here* is omitted.

Formally, a (cell-like) P system is a construct

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, l_{in}, l_{out})$$

where O is the alphabet of objects, μ is the membrane structure (with m membranes), $w_1, \dots, w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \dots, R_m$ are finite sets of evolution rules, associated with the regions of μ, l_{in} is the label of the membrane region where the inputs are put at the beginning of a computation, and l_{out} indicates the region from which the outputs are taken; l_{out}/l_{in} being 0 indicates that the output/input is taken from the environment.

If a rule $u \to v$ has $|u| > 1$, then it is called *cooperative* (abbreviated *coo*); otherwise, it is called *non-cooperative* (abbreviated *ncoo*). In *catalytic P systems* non-cooperative as well as *catalytic rules* of the form $ca \to cv$ are used, where c is a *catalyst* – a special object that never evolves and never passes through a membrane, but it just assists object a to evolve to the multiset v. In a *purely catalytic P system* only catalytic rules are allowed. In both catalytic and purely catalytic P systems, in their description O is replaced by O, C in order to specify those objects from O that are the catalysts in the set C.

The evolution rules are used in the non-deterministic maximally parallel way, i.e., in any computation step of Π a multiset of rules is chosen from the sets $R_1, \dots, R_m$ in such a way that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the membrane regions $1, \dots, m$. A *configuration* of a system is given by the membranes and the objects present in the compartments of the system. Starting from a given *initial configuration* and applying evolution rules as described above, we get *transitions* among configurations; a sequence of transitions forms a computation. A computation is *halting* if it reaches a configuration where no rule can be applied any more.

In the *generative case*, a halting computation has associated a result, in the form of the number of objects present in membrane l_{out} in the halting configuration (l_{in} can be omitted). The set of non-negative integers and the set of (Parikh) vectors of non-negative integers obtained as results of halting computations in Π are denoted by $N_{gen}(\Pi)$ and $Ps_{gen}(\Pi)$, respectively.

In the *accepting case*, for $l_{in} \neq 0$, we accept all (vectors of) non-negative integers whose input, given as the corresponding numbers of objects in membrane l_{in}, leads to a halting computation (l_{out} can be omitted); the set of non-negative integers and the set of (Parikh) vectors of non-negative integers accepted in that way by halting computations in Π are denoted by $N_{acc}(\Pi)$ and $Ps_{acc}(\Pi)$, respectively.

For the input being taken from the environment, i.e., for $l_{in} = 0$, we need an additional target indication *come*; $(a, come)$ means that the object a is taken into the skin from the environment (all objects there are assumed to be available in an unbounded number). The multiset of all objects taken from the environment during a halting computation then is the multiset accepted by this accepting P system, which in this case we shall call a *P automaton* [3]; the set of non-negative integers and the set of (Parikh) vectors of non-negative integers accepted by halting computations in Π are denoted by $N_{aut}(\Pi)$ and $Ps_{aut}(\Pi)$, respectively.

A P system Π can also be considered as a system computing a partial recursive function (in the deterministic case) or even a partial recursive relation (in the non-deterministic case), with the input being given in a membrane region $l_{in} \neq 0$ as in the accepting case or being taken from the environment as in the automaton case. The corresponding functions/relations computed by halting computations in Π are denoted by $ZY_{\alpha}(\Pi)$, $Z \in \{Fun, Rel\}$, $Y \in \{N, Ps\}$, $\alpha \in \{acc, aut\}$.

Computational completeness for (generating) catalytic P systems can be achieved when using two catalysts or with three catalysts in purely catalytic P systems, and the same number of catalysts is needed for P automata; in accepting P systems, the number of catalysts increases with the number of components in the vectors of natural numbers to be analyzed [7]. It is a long-time open problem how to characterize the families of sets of (vectors of) natural numbers generated by (purely) catalytic P systems with only one (two) catalysts. Using additional control mechanisms as, for example, priorities or promoters/inhibitors, P systems with only one (two) catalyst(s) can be shown to be computationally complete, e.g., see Chapter 4 in [17]. Last year several other variants of control mechanism have been shown to lead to computational completeness in (purely) catalytic P systems using only one (two) catalyst(s), see [6], [9], and [10]. In this paper we are going to investigate the power of using matter/antimatter annihilation rules – with the astonishing result, that no catalysts are needed any more in case the annihilation rules have weak priority over the other rules.

The family of sets $Y_{\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, computed by P systems with at most m membranes and cooperative rules and with non-cooperative rules is denoted by $Y_{\delta}OP_m(coo)$ and $Y_{\delta}OP_m(ncoo)$, respectively. The family of sets $Y_{\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, computed by (purely)

catalytic P systems with at most m membranes and at most k catalysts is denoted by $Y_\delta OP_m\,(cat_k)$ ($Y_\delta OP_m\,(pcat_k)$). The following characterizations are known:

Theorem 1. *For any $m \geq 1$ and any $Y \in \{N, Ps\}$,*

$$YREG = Y_{gen}OP_m\,(ncoo) \subset Y_{gen}OP_m\,(coo) = YRE.$$

Theorem 2. *For any $m \geq 1$, $d \geq 1$, $\delta \in \{gen, aut\}$,*

$$Ps_{acc}OP_m\,(cat_{d+2}) = Ps_{acc}OP_m\,(pcat_{d+3}) = N^dRE.$$
$$Ps_\delta OP_m\,(cat_2) = Ps_\delta OP_m\,(pcat_3) = PsRE.$$

4 Using Matter and Anti-Matter

This concept to be used in (catalytic) P systems is a direct generalization of the idea of anti-spikes from spiking neural P systems (see [14]): for each object a we introduce the anti-matter object a^-. We can look at these anti-matter objects a^- as objects of their own or else we may extend the notion of a (finite) multiset over the (finite) alphabet V, $V = \{a_1, \cdots, a_n\}$, as a mapping $f : V \longrightarrow \mathbb{N}$ to a mapping $f : V \longrightarrow \mathbb{Z}$ now also allowing negative values. In a usual way, such an extended multiset on $\mathbb{Z}$ is represented by $\left\langle a_1^{f(a_1)}, \cdots, a_n^{f(a_n)} \right\rangle$. A unique string representation for such an extended multiset is obtained by assigning a string over the (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_n, {a_n}^- \rangle$ as ${a_1}^{f(a_1)} \cdots {a_n}^{f(a_n)}$ such that $(a_i)^{-m}$, $m > 0$, is represented by $({a_i}^-)^m$, $1 \leq i \leq n$. Any other string having the same Parikh vector with respect to the (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_n, {a_n}^- \rangle$ can be used for representing the multiset given by f as well.

As in spiking neural P systems with anti-spikes, also in cell-like P systems we might consider the annihilation of matter and anti-matter objects to happen in zero-time or in an intermediate step between normal derivation steps in the maximally parallel mode. Whenever related matter a and anti-matter a^- meet, they annihilate each other, as, for example, in an extended multiset on $\mathbb{Z}$ matter a and anti-matter a^- cannot exist at the same moment, hence, also not in a string representing an extended multiset on $\mathbb{Z}$. Yet in this paper we consider both objects and anti-objects to be handled by usual evolution rules; the annihilation of matter and anti-matter objects then corresponds to an application of the (non-context-free!) rule $aa^- \to \lambda$. (Purely) catalytic P systems thus can be extended to

- allow for annihilation rules of the form $aa^- \to \lambda$ (and for catalytic annihilation rules $caa^- \to c$, where c is a catalyst) over an (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_n, {a_n}^- \rangle$, *or*
- to work on extended multisets over the (ordered) alphabet $\langle a_1, \cdots, a_n \rangle$.

In contrast to the case described above, now in an instantaneous description of a configuration of a P system both matter and anti-matter objects may appear. When working with context-free or catalytic rules over an (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_n, {a_n}^- \rangle$, we may give the matter/anti-matter annihilation rules

weak priority over all other rules – in order to not have matter a and anti-matter a^- in some configuration at the same moment and let them "survive" for longer.

We now consider catalytic P systems extended by also allowing for annihilation rules $aa^- \to \lambda$, with these rules having weak priority over all other rules, i.e., other rules can only be applied if no annihilation rule could still bind the corresponding objects. The family of sets $Y_\delta(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, and the family of functions/relations $ZY_\alpha(\Pi)$, $Z \in \{Fun, Rel\}$, $\alpha \in \{acc, aut\}$, computed by such extended P systems with at most m membranes and k catalysts is denoted by $Y_\delta OP_m(cat(k), antim/pri)$ and $ZY_\alpha OP_m(cat(k), antim/pri)$; we omit $/pri$ for the families without priorities.

The matter/anti-matter annihilation rules are so powerful that we only need the minimum number of catalysts, i.e., zero!

Theorem 3. *For any $n \geq 1$, $k \geq 0$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$Y_\delta OP_n(cat(k), antim/pri) = YRE \text{ and}$$
$$ZY_\alpha OP_n(cat(k), antim/pri) = ZYRE.$$

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. We now construct a one-membrane P system, initially containing only the object l_0, which simulates M. The contents of register r is represented by the number of copies of the object a_r, $1 \leq r \leq m$, and for each object a_r we also consider the corresponding anti-object ${a_r}^-$. The instructions of M are simulated as follows:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Simulated by the rules

  $$l_1 \to a_r l_2 \text{ and } l_1 \to a_r l_3.$$

- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$.
  As rules common for all simulations of SUB-instructions, we have

  $${a_r}^- \to \#^-, \ 1 \leq r \leq m,$$

  and the annihilation rules

  $$a_r {a_r}^- \to \lambda, \ 1 \leq r \leq m, \text{ and } \#\#^- \to \lambda$$

  as well as the trap rules

  $$\#^- \to \#\# \text{ and } \# \to \#\#;$$

  these last two rules lead the system into an infinite computation whenever a trap symbol is left without being annihilated.
  The *zero test* for instruction l_1 is simulated by the rules

  $$l_1 \to {l_1}' {a_r}^- \text{ and } {l_1}' \to \# l_3.$$

The symbol # generated by the second rule $l_1' \to \#l_3$ can only be eliminated if the anti-matter ${a_r}^-$ generated by the first rule $l_1 \to l_1' {a_r}^-$ is not annihilated by a_r, i.e., only if register r is empty.
The *decrement case* for instruction l_1 is simulated by the rule

$$l_1 \to l_2 {a_r}^-.$$

The anti-matter ${a_r}^-$ either correctly annihilates one matter a_r thus decrementing the register r or else traps an incorrect guess by forcing the symbol ${a_r}^-$ to evolve to $\#^-$ and then to $\#\#$ in the next two steps in case register r is empty.

- $l_h : HALT$. Simulated by $l_h \to \lambda$.
When the computation in M halts, the object l_h is removed, and no further rules can be applied provided the simulation has been carried out correctly, i.e., if no trap symbols # are present in this situation. The remaining objects in the system represent the result computed by M. □

Without this priority of the annihilation rules, the construction is not working, hence, a characterization of the families $Y_\delta OP_n\,(ncoo, antim)$ as well as $ZY_\alpha OP_n\,(ncoo, antim)$ remains as an open problem. Yet in addition using catalytic rules with one catalyst again allows us to obtain computational completeness:

Theorem 4. *For any $n \geq 1$, $k \geq 1$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$Y_\delta OP_n\,(cat(k), antim) = YRE \qquad \textit{and}$$
$$ZY_\alpha OP_n\,(cat(k), antim) = ZYRE.$$

Proof. We again consider a register machine $M = (m, B, l_0, l_h, P)$ as in the previous proof, and construct the catalytic P system

$$\Pi = (O, \{c\}, [\ ]_1, cl_0, R_1, l_{in}, 1) \text{ with}$$
$$O = \{a_r, {a_r}^- \mid 1 \leq r \leq m\} \cup \{l, l', l'' \mid l \in B\} \cup \{\#, \#^-, d\},$$

with the single catalyst c in the skin membrane. The results now are sent to the environment, in order not to have to count the catalyst in the skin membrane; for that purpose, we simply use the rule $a_i \to (a_i, out)$ for the output symbols a_i (we assume that output registers of M are only incremented).

For each ADD-instruction $l_1 : (ADD\,(j), l_2, l_3)$ in P, we again take the rules

$$l_1 \to a_r l_2 \text{ and } l_1 \to a_r l_3.$$

For each SUB-instruction $l_1 : (SUB\,(r), l_2, l_3)$, we now consider the four rules

$$l_1 \to l_2 {a_r}^-,$$
$$l_1 \to l_1'' d {a_r}^-,$$
$$l_1'' \to l_1', \text{ and}$$
$$l_1' \to \#l_3.$$

As rules common for all SUB-instructions, we again add the matter/antimatter annihilation rules

$$a_r {a_r}^- \to \lambda \text{ and } \#\#^- \to \lambda$$

as well as the trap rules

$$\# \to \#\# \text{ and } \#^- \to \#\#,$$

but in addition, also

$$d \to \#\#$$

as well as the catalytic rules

$$cd \to c \text{ and } c{a_r}^- \to c\#^-,\ 1 \leq r \leq m.$$

The decrement case is simulated as in the previous proof, by using the rule $l_1 \to l_2 {a_r}^-$ and then applying the annihilation rule $a_r {a_r}^- \to \lambda$. The zero-test now is initiated with the rule $l_i \to l_i'' d {a_r}^-$ thus introducing the (dummy) symbol d which keeps the catalyst busy for one step, where the catalytic rule $cd \to c$ has to be applied in order to avoid the application of the trap rule $d \to \#\#$. If register r is empty, then ${a_r}^-$ cannot be annihilated and therefore evolves to $\#^-$ in the third step by the application of the catalytic rule $c{a_r}^- \to c\#^-$, which symbol $\#^-$ afterwards annihilates the symbol $\#$ generated by the rule $l_i' \to \# l_k$ in the same step; if register r is not empty, ${a_r}^-$ is annihilated by some copy of a_r already in the first step, hence, the trap symbol $\#$ generated by the rule $l_i' \to \# l_k$ does not find its anti-matter $\#^-$ and therefore evolves to $\#\#$, thus leading to an infinite computation. Altough the annihilation rule $a_r {a_r}^- \to \lambda$ now does not have priority over the catalytic rule $c{a_r}^- \to c\#^-$, maximal parallelism enforces $a_r {a_r}^- \to \lambda$ to be applied, if possible, already in the first step instead of $c{a_r}^- \to c\#^-$, as in a successful derivation the catalyst c first has to eliminate the dummy symbol d.

The rule $l_h \to \lambda$ is applied at the end of a successful simulation of the instructions of the register machine M, and the computation halts if no trap symbol $\#$ is present; the symbols sent out to the environment during the computation represent the result of this halting computation. □

In the accepting case, with priorities, we can even simulate the actions of a deterministic register machine in a deterministic way, i.e., for each configuration of the system, there can be at most one multiset of rules applicable to it.

Theorem 5. *For any $n \geq 1$, $k \geq 0$, and $Y \in \{N, Ps\}$,*

$$Y_{detacc}OP_n\,(cat(k), antim/pri) = YRE \text{ and}$$
$$FunY_{detacc}OP_n\,(cat(k), antim/pri) = FunYRE.$$

Proof. We only show how the SUB-instructions of a register machine $M = (m, B', l_0, l_h, P)$ can be simulated in a deterministic way without introducing a trap symbol and therefore causing infinite loops by them:

Let $B = \{l \mid l : (SUB\,(r)\,, l', l'') \in P\}$ and, for every register r,

$$\tilde{M}_r = \left\{\tilde{l} \mid l : (SUB\,(r)\,, l', l'') \in P\right\}, \quad \hat{M}_r = \left\{\hat{l} \mid l : (SUB\,(r)\,, l', l'') \in P\right\},$$
$$\tilde{M}_r{}^- = \left\{\tilde{l}^- \mid l : (SUB\,(r)\,, l', l'') \in P\right\}, \hat{M}_r{}^- = \left\{\hat{l}^- \mid l : (SUB\,(r)\,, l', l'') \in P\right\}.$$

We now take the rules

$$a_r{}^- \to \tilde{M}_r{}^- \hat{M}_r$$

and the annihilation rules $a_r a_r{}^- \to \lambda$ for every register r as well as $\hat{l}\hat{l}^- \to \lambda$ and $\tilde{l}\tilde{l}^- \to \lambda$ for all $l \in B$. Then a SUB-instruction $l_1 : (SUB\,(r)\,, l_2, l_3)$, with $l_1 \in B$, $l_2, l_3 \in B'$, $1 \le r \le m$, is simulated by the rules

$$\begin{aligned}
&l_1 \to \bar{l}_1 a_r{}^-,\\
&\bar{l}_1 \to \hat{l}_1{}^- (\tilde{M}_r \setminus \{\tilde{l}_1\}),\\
&\hat{l}_1{}^- \to l_2 (\tilde{M}_r{}^- \setminus \{\tilde{l}_1{}^-\}), \text{ and}\\
&\tilde{l}_1{}^- \to l_3 (\hat{M}_r{}^- \setminus \{\hat{l}_1{}^-\}).
\end{aligned}$$

The symbol $\hat{l}_1{}^-$ generated by the second rule is eliminated again and replaced by $\tilde{l}_1{}^-$ if $a_r{}^-$ is not annihilated (which indicates that the register is empty). □

5 Small Universal P Systems with Anti-Matter

In [12], several variants of universal register machines were exploited. The main interesting variant for the results presented in this paper is shown in Figure 1.

In the diagram of the universal register machine U_{32} in Figure 1, the operations used on the registers are: the *zero-test* on register i is indicated by a rhomboid inclosing the encryption Ri, and in the case that the contents of register i is zero, the next operation is the one to be reached with the arc labeled by z; the *increment* operation is depicted by a rectangle with the encryption RiP, and the *decrement* operation by a rectangle with the encryption RiM (as the decrement operation RiM is always preceded by the corresponding zero-test, it can always be carried out). The states are depicted directly at the corresponding operations; q_1 is the initial state, and the state where the U_{32} stops is indicated by $STOP$ in Figure 1.

Remark 1. The universal register machine U_{32} uses a very sophisticated number-theoretic encoding of the enumeration of a specific variant of register machines. The code of the register machine to be simulated is put into register 1, the input number into register 2 (where also the output will be computed). The instructions are decoded in the *Instruction reader* part and the *Decoder* part (which essentially performs a division by three), and these instructions work on the registers 0, 2, and 3 (as we know, e.g., see [13], three registers are sufficient to simulate any other register machine). Thus, U_{32} can compute any partial recursive function $f : \mathbb{N} \to \mathbb{N}$ (with input and output number in register 2) in the same way as the register machine encoded by the number in register 1 computes f. For the following

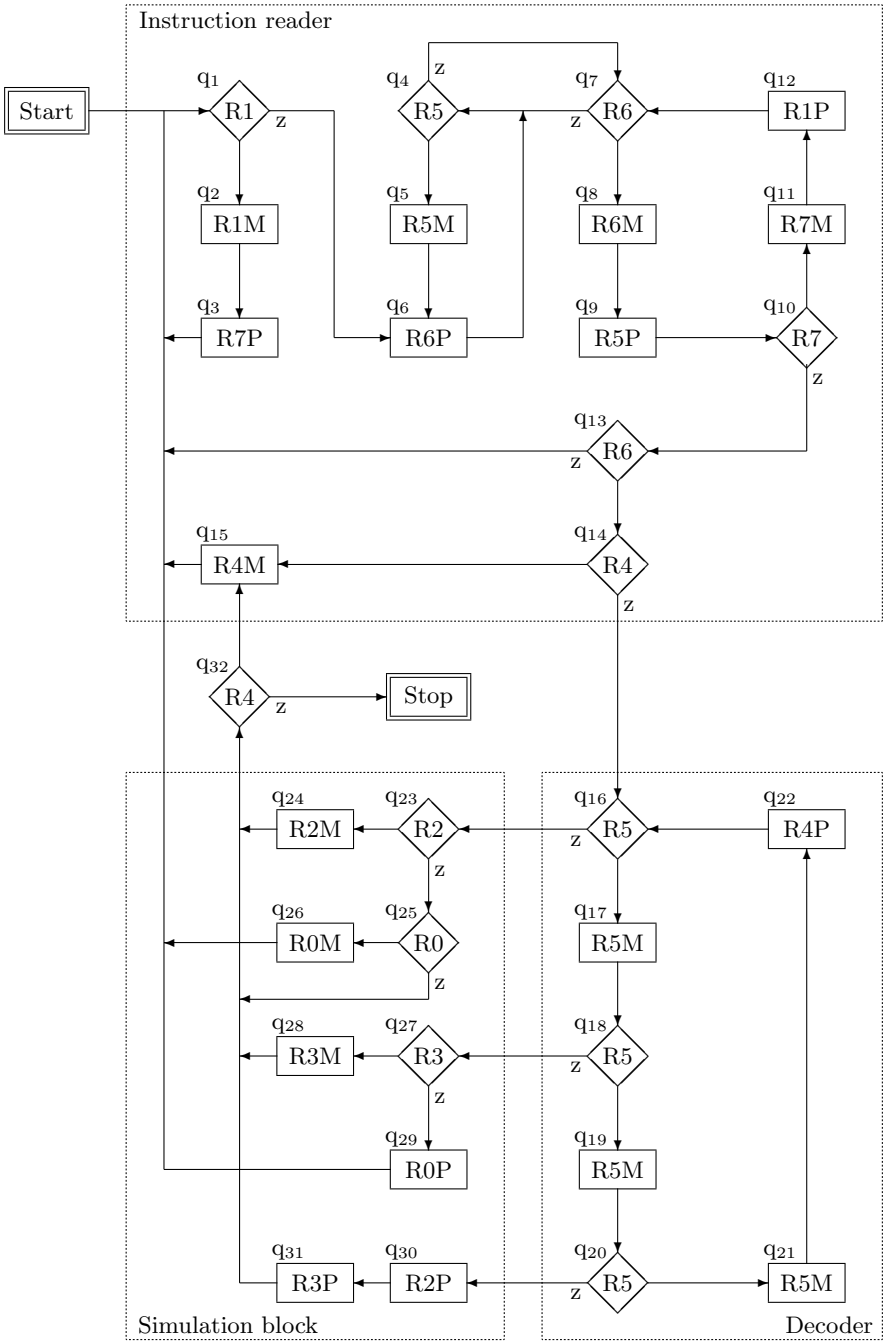


Fig. 1. The universal register machine U_{32}.

constructions it is important to note that U_{32} only stops when having finished the simulation of the register machine encoded in register 1 with the input in register 2, but enters an infinite computation otherwise.

A thorough analysis of the universal register machine U_{32} shows that when it halts not only register 2 as the output register, but also register 6 and register 1 (still containing the code of the register machine to be simulated) may be non-empty. On the other hand, due to the features of the 3-register machine as constructed in [13] and simulated by U_{32} in registers 0, 2, and 3, at the end of a computation registers 0 and 3 are empty (observe that the emptiness of these registers cannot be inferred from the program of U_{32}); in the accepting case, register 2 is empty, too.

In order to produce better descriptional complexity results with respect to the number of rules than those we would immediately get when applying the constructions given in the proof of Theorem 3, we introduce a generalization of register machines or counter automata.

Generalized counter automata.

For a register machine $M = (m, B, l_0, l_h, P)$ consider the more general type of instructions $i : (q, M_-, N, M_+, q')$, where $q, q' \in Q$ are states, $N \subseteq R$ is a set of registers, and M_-, M_+ are multisets of registers. Such a register machine applies instruction i as follows: first, multiset M_- is subtracted from the register values (i.e., for each register $j \in R$, $M_-(j)$ is subtracted from the contents of register j; if at least one resulting value would be negative, the machine is blocked without producing any result); second, the subset N of registers is checked to be zero (if at least one of them is found to be non-zero, the machine is blocked without producing any result); third, the multiset M_+ is added to the register values (i.e., for each register $j \in R$, $M_+(j)$ is added to the contents of register j), and finally the state changes to q'.

The work of such a register machine, now also called a *generalized counter automaton* and written $M = (m, B, l_0, q_h, P)$, consists of derivation steps applying instructions, chosen in a non-deterministic way, associated with the current state. The computation starts in the initial state q_0, and we say that it halts if the final state q_h has been reached (which replaces the condition of reaching the final HALT-instruction labeled by l_h).

Theorem 6. *There exist small universal P systems with non-cooperative rules and matter/anti-matter annihilation rules – with* 9 *annihilation rules and, in total,* 52 *rules in the* accepting *case,* 59 *rules in the* generating *case, and* 57 *rules in the* computing *case.*

Proof. We start with a translation of the P system from Theorem 4 in [8] (obtained from the universal register machine U_{32} machine in [12]). This sequential

antiport P system with forbidden contexts can be written with the instructions of a generalized counter automaton as follows:

$1 : (q_1, \langle 1\rangle, \{\}, \langle 7\rangle, q_1)$,
$2 : (q_1, \langle\rangle, \{1\}, \langle 6\rangle, q_4)$,
$3 : (q_4, \langle 5\rangle, \{\}, \langle 6\rangle, q_4)$,
$4 : (q_4, \langle 6\rangle, \{5\}, \langle 5\rangle, q_{10})$,
$5 : (q_{10}, \langle 7, 6\rangle, \{\}, \langle 1, 5\rangle, q_{10})$,
$6 : (q_{10}, \langle 7\rangle, \{6\}, \langle 1\rangle, q_4)$,
$7 : (q_{10}, \langle\rangle, \{6, 7\}, \langle\rangle, q_1)$,
$8 : (q_{10}, \langle 6, 4\rangle, \{7\}, \langle\rangle, q_1)$,
$9 : (q_{10}, \langle 6, 5\rangle, \{7, 4\}, \langle\rangle, q_{18})$,
$10 : (q_{18}), \langle 5^3\rangle, \{\}, \langle 4\rangle, q_{18})$,
$11 : (q_{18}, \langle\rangle, \{5, 3\}, \langle 0\rangle, q_1)$,
$12 : (q_{18}, \langle 5^2, 0\rangle, \{5, 2\}, \langle\rangle, q_1)$,
$13 : (q_{18}, \langle 5^2, 2\rangle, \{5\}, \langle\rangle, q_1)$,
$14 : (q_{18}, \langle 5^2\rangle, \{5, 2, 0\}, \langle\rangle, q_1)$
$15 : (q_{18}, \langle 3, 4\rangle, \{5\}, \langle\rangle, q_1)$,
$16 : (q_{18}, \langle 5, 4\rangle, \{5\}, \langle 2, 3\rangle, q_1)$.

The system constructed in [8] halts when none of the rules 10 to 16 can be applied any more. For halting with the generalized counter automaton we have to define a halting state q_h and instructions how to reach this halting state. The details of this halting in the generalized counter automaton and how to halt in the P system to be constructed in the following, depending on the mode the automaton is used for – generating, accepting, computing – will be discussed later in the proof. We first show how these 16 instructions of the generalized counter automaton listed above can be simulated by a P system with anti-matter.

For a generalized counter automaton $M = (m, B, l_0, q_h, P)$, let

$$k = 1 + \max_{i:(q,M_-,N,M_+,q')\in P}(|M_-|, |N|).$$

We consider the following rules (common for different instructions of M):

$$\#^- \to \#^k,\ \# \to \#^k,\ \#\#^- \to \lambda,\ a_r \to \#^-,\ a_r a_r^- \to \lambda,\ r \in R.$$

Now we present the simulation of instruction $i : (q, M_-, N, M_+, q') \in P$. First we consider the case when M_- and N have no common elements, and moreover, we also assume that M_- does not overlap with M_+ (otherwise such an instruction can be split into two instructions; notice that this condition is already satisfied in the rules given above).

$$q \to l_i \prod_{r\in N} {a_r}^-,\ l_i \to q'(\prod_{r\in N} \#)(\prod_{r\in M_-} {a_r}^-) \prod_{r\in M_+} a_r.$$

Indeed, the zero-test is successful if *none* of the objects a_r^- generated in the first step annihilates with the corresponding register symbols a_r; they have to change into objects $\#^-$ to annihilate with the same number of objects # produced in the next step. The decrement is successful if *all* objects ${a_r}^-$ generated in the second step annihilate with the corresponding register symbols a_r. If either decrement or zero-test fail, then at least either one # or one $\#^-$ will be produced without its annihilation partner, leading to producing objects # in a geometric progression, ensuring that such computations do not produce any result (notice that no objects # or $\#^-$ are produced in the first step of the simulation of any instruction).

If the zero-test set N is empty, then the first step is a simple renaming, and thus can be combined with the second step, yielding just one rule

$$q \to q'(\prod\nolimits_{r\in M_-} {a_r}^-) \prod\nolimits_{r\in M_+} a_r.$$

Clearly, if M_- and N overlap, such an instruction can be broken down into two subsequent instructions of the generalized counter automaton. However, a more efficient solution with only three rules exists:

$$q \to l_i \prod\nolimits_{r\in M_-} {a_r}^-,\ l_i \to l_i' \prod\nolimits_{r\in N} {a_r}^-,\ l_i' \to q(\prod\nolimits_{r\in N} \#^-) \prod\nolimits_{r\in M_+} a_r.$$

The direct translation of the instructions of the generalized counter automaton given in the table at the beginning of the proof yields the following rules:

$$\begin{array}{lllll}
1: & q_1 \to q_1 {a_1}^- a_7, & & & \\
2: & q_1 \to l_2 {a_1}^-, & l_2 \to q_4 \# a_6, & & \\
3: & q_4 \to q_4 {a_5}^- a_6, & & & \\
4: & q_4 \to l_4 {a_5}^-, & l_4 \to q_{10} \# {a_6}^- a_5, & & \\
5: & q_{10} \to q_{10} {a_7}^- {a_6}^- a_1 a_5, & & & \\
6: & q_{10} \to l_6 {a_6}^-, & l_6 \to q_4 \# {a_7}^- a_1, & & \\
7: & q_{10} \to l_7 {a_6}^- {a_7}^-, & l_7 \to q_1 \#\#, & & \\
8: & q_{10} \to l_8 {a_7}^-, & l_8 \to q_1 \# {a_6}^- {a_4}^-, & & \\
9: & q_{10} \to l_9 {a_7}^- {a_4}^-, & l_9 \to q_{18} \#\# {a_6}^- {a_5}^-, & & \\
10: & q_{18} \to q_{18} {a_5}^- {a_5}^- {a_5}^- a_4, & & & \\
11: & q_{18} \to l_{11} {a_5}^- {a_3}^-, & l_{11} \to q_1 \#\# a_0, & & \\
12: & q_{18} \to l_{12} {a_5}^- {a_5}^- a_0^-, & l_{12} \to l_{12}' {a_5}^- {a_2}^-, & l_{12}' \to q_1 \#\#, & \\
13: & q_{18} \to l_{13} {a_5}^- {a_5}^- {a_2}^-, & l_{13} \to l_{13}' {a_5}^-, & l_{13}' \to q_1 \#, & \\
14: & q_{18} \to l_{14} {a_5}^- {a_5}^-, & l_{14} \to l_{14}' {a_5}^- {a_2}^- {a_0}^-, & l_{14}' \to q_1 \#\#\#, & \\
15: & q_{18} \to l_{15} {a_5}^-, & l_{15} \to q_1 \# {a_3}^- {a_4}^-, & & \\
16: & q_{18} \to l_{16} {a_5}^- {a_4}^-, & l_{16} \to l_{16}' {a_5}^-, & l_{16}' \to q_1 \# a_2 a_3, &
\end{array}$$

In the *accepting case*, the input/output register 2 as well as the registers 0 and 3 are empty (see Remark 1); in order to reach the STOP in Figure 1, starting from q_{10}, register 6 must be non-empty, but registers 4, 5, 2, and 0 must be empty. Hence, we add the additional rule

$$17 : (q_{10}, \langle 6 \rangle, \{4, 5, 2, 0\}, \langle\rangle, q_h),$$

which is simulated by the rules

$$17 : q_{10} \to l_{17} {a_4}^- {a_5}^- {a_2}^- {a_0}^-,\ l_{17} \to q_h \#\#\#\# {a_6}^-.$$

As the rules with l_7 and l_{12}' on the left side have the same right side, we can replace l_{12}' by l_7, thus decreasing the number of non-cooperative rules by one.

On the other hand, we have to add the rules

$$a_r \to \#^-,\ 0 \le r \le 7,$$

and the trap rules

$$\#^- \to \#^5 \text{ and } \# \to \#^5$$

as well as the annihilation rules

$$(\#\#^- \to \lambda) \text{ and } (a_r a_r{}^- \to \lambda),\ 0 \le r \le 7.$$

In sum, we obtain the universal P system with anti-matter

$$\begin{aligned}&\Pi = (O, [\ ]_1, q_1, R_1, 1) \text{ where}\\ &O = \{l_2, l_4, l_6, l_7, l_8, l_9, l_{11}, l_{12}, l_{13}, l'_{13}, l_{14}, l'_{14}, l_{15}, l_{16}, l'_{16}, l_{17}\}\\ &\quad \cup \{q_1, q_4, q_{10}, q_{18}, q_h\} \cup \{a, a^- \mid a \in \{a_j \mid 0 \le j \le 7\} \cup \{\#\}\}\end{aligned}$$

with the rules in R_1 as described above; in total, the number of rules is 52, where 9 of them are the model-defined annihilation rules.

For the *computing case*, we have to re-introduce the state q_{32} of U_{32}, i.e., we replace instructions 15, 16, and 17 in the table of instructions for the generalized counter automaton by the instructions 15′, 16′, 17′, and 18′; moreover we have to "clean" registers 1 and 6 (see Remark 1) and therefore to add the state q'_h and another three instructions:

$$\begin{aligned}&15' : (q_{18}, \langle 3\rangle, \{5\}, \langle\rangle, q_{32}),\\ &16' : (q_{18}, \langle 5\rangle, \{5\}, \langle 2, 3\rangle, q_{32}),\\ &17' : (q_{32}, \langle 4\rangle, \{\}, \langle\rangle, q_1),\\ &18' : (q_{32}, \langle\rangle, \{4\}, \langle\rangle, q'_h),\\ &19' : (q'_h, \langle 1\rangle, \{\}, \langle\rangle, q'_h),\\ &20' : (q'_h, \langle 6\rangle, \{\}, \langle\rangle, q'_h),\\ &21' : (q'_h, \langle\rangle, \{1, 6\}, \langle\rangle, q_h).\end{aligned}$$

These instructions can be simulated by the following rules:

$$\begin{aligned}&15' : q_{18} \to l_{15} a_5{}^-, \quad l_{15} \to q_{32} \# a_3{}^-,\\ &16' : q_{18} \to l_{16} a_5{}^-, \quad l_{16} \to l'_{16} a_5{}^-, \quad l'_{16} \to q_{32} \# a_2 a_3,\\ &17' : q_{32} \to q_1 a_4{}^-,\\ &18' : q_{32} \to l_{18} a_4{}^-, \quad l_{18} \to q'_h \#,\\ &19' : q'_h \to q'_h a_1{}^-,\\ &20' : q'_h \to q'_h a_6{}^-,\\ &21' : q'_h \to q_h a_1{}^- a_6{}^-,\ q_h \to \#\#.\end{aligned}$$

In that way, we obtain a universal P system Π' with anti-matter having 57 rules, i.e., 49 non-cooperative rules and 9 model-defined annihilation rules:

$$\begin{aligned}&\Pi' = (O', [\ ]_1, q_1, R'_1, 1, 1) \text{ where}\\ &O' = \{l_2, l_4, l_6, l_7, l_8, l_9, l_{11}, l_{12}, l_{13}, l'_{13}, l_{14}, l'_{14}, l_{15}, l_{16}, l'_{16}, l_{18}\}\\ &\quad \cup \{q_1, q_4, q_{10}, q_{18}, q_{32}, q'_h, q_h\} \cup \{a, a^- \mid a \in \{a_j \mid 0 \le j \le 7\} \cup \{\#\}\}\end{aligned}$$

and R'_1 contains the rules 1 to 14, 15′ to 21′ as well as the rules $a_r \to \#^-$, $0 \le r \le 7$, the trap rules $\#^- \to \#^4$ and $\# \to \#^4$ as well as the annihilation rules

$(\#\#^- \to \lambda)$ and $(a_r a_r{}^- \to \lambda)$, $0 \leq r \leq 7$. The P system now halts with the skin membrane only containing copies of the symbol a_2 representing the output value.

In fact, this construction could also be used for the accepting case, where we could already stop with q'_h, as we need not "clean" registers 1 and 6; in total this would yield only one rule more, i.e., 53 instead of 52 rules.

Finally, in the *generating case*, we start with the new initial state q_0 and add the two rules $q_0 \to a_2 q_0$ and $q_0 \to q_1$, which allows us to produce, in a non-deterministic way, an input for U_{32} simulating the identity function on the domain of the set to be generated by the P system, i.e., we get a P system with 59 rules. □

6 When Matter/Anti-Matter Annihilation Generates Energy

The matter/anti-matter annihilation may also be assumed to result in the generation of a specific amount of "energy", which is also well motivated by physics. In the definitions of these systems, the matter/anti-matter annihilation rules $a_r a_r{}^- \to \lambda$ are replaced by $a_r a_r{}^- \to e$ where e is a symbol denoting this special amount of energy.

The family of sets $Y_\delta(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, and the set of functions/relations $ZY_\alpha(\Pi)$, $Z \in \{Fun, Rel\}$, $\alpha \in \{acc, aut\}$, computed by such P systems with at most m membranes and k catalysts is denoted by $Y_\delta OP_m(cat(k), antimen/pri)$ and $ZY_\alpha OP_m(cat(k), antimen/pri)$; we omit $/pri$ for the families without priorities.

The following results are immediate consequences of the corresponding Theorems 3 and 5 – in both cases, each matter/anti-matter annihilation rule $xx^- \to \lambda$ is replaced by $xx^- \to e$ where e is this symbol denoting a special amount of energy, and, in addition, we add the rule $e \to \lambda$:

Corollary 1. *For any $n \geq 1$, $k \geq 0$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$Y_\delta OP_n(cat(k), antimen/pri) = YRE \text{ and}$$
$$ZY_\alpha OP_n(cat(k), antimen/pri) = ZYRE.$$

Corollary 2. *For any $n \geq 1$, $k \geq 0$, and $Y \in \{N, Ps\}$,*

$$Y_{detacc} OP_n(cat(k), antimen/pri) = YRE \text{ and}$$
$$FunY_{detacc} OP_n(cat(k), antimen/pri) = FunYRE.$$

But we can even show more, i.e., omitting the rule $e \to \lambda$ and leaving the amount of energy represented by the number of copies of e in the system, the energy inside the system at the end of a successful computation is a direct measure for the number of SUB-instructions simulated by the P system or even a measure for the number of all instructions which were simulated.

Corollary 3. *The construction in the proof of Theorem 3 can be adapted in such a way that the simulation of each instruction of the register machine takes exactly three steps (including the annihilation rules), and moreover, the number of energy objects e at the end of a successful computation exactly equals the number of instructions simulated.*

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. Following the construction given in the proof of Theorem 3, the instructions of M now can be simulated as follows:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Simulated by the rules
  $$l_1 \to {l_1}',$$
  $${l_1}' \to {l_1}'',$$
  $${l_1}'' \to e a_r l_2,$$
  $${l_1}'' \to e a_r l_3.$$
- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$.
  As rules common for all simulations of SUB-instructions, we have
  $$a_r{}^- \to \#^-,\ 1 \leq r \leq m,$$
  $$a_r a_r{}^- \to e,\ 1 \leq r \leq m,$$
  $$\#\#^- \to e,$$
  $$\#^- \to \#\#,$$
  $$\# \to \#\#.$$
  The *zero test* for instruction l_1 is simulated by the rules
  $$l_1 \to {l_1}' a_r{}^-,$$
  $${l_1}' \to \# {l_1}'', \text{ and}$$
  $${l_1}'' \to l_3;$$
  - the symbol # generated by the second rule ${l_1}' \to \# {l_1}''$ can only be eliminated if the anti-matter $a_r{}^-$ generated by the first rule $l_1 \to {l_1}' a_r{}^-$ is not annihilated by a_r, i.e., only if register r is empty; e is generated by $\#\#^- \to e$.

  The *decrement case* for instruction l_1 is simulated by the rules
  $$l_1 \to \tilde{l}_1 {a_r}^-,$$
  $$\tilde{l}_1 \to \tilde{l}_1',$$
  $$\tilde{l}_1' \to l_2;$$
  - here, e is generated by $a_r a_r{}^- \to e$.
- $l_h : HALT$. Simulated by the rules
  $$l_h \to {l_h}',$$
  $${l_h}' \to {l_h}'',$$
  $${l_h}'' \to e.$$

In each case, exactly one symbol e is generated during each cycle of three steps simulating an instruction of M. □

Remark 2. Let M be a register machine and

$$RS(M) = \{(n, m) \mid n \in L(M), n \text{ is computed by } M \text{ in } m \text{ steps}\}.$$

Then, according to [2], RS is recursive. Hence, although $L(M)$ may not be recursive, $RS(M)$ is recursive in any case.

Now let $L \in NRE$ and $L = L(M)$ for a register machine M. Following the construction given in the proof of Corollary 3, we can construct a P system with energy Π such that $Ps(\Pi) = RS(M)$.

7 Computing with Integers

As already discussed in Section 4, given an alphabet $V = \{a_1, \cdots, a_d\}$ we may extend the notion of a (finite) multiset over V as a mapping $f : V \longrightarrow \mathbb{N}$ to a mapping $f : V \longrightarrow \mathbb{Z}$ now also allowing negative values, with a unique string representation for such an extended multiset being obtained by assigning a string over the (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_d, {a_d}^- \rangle$ as ${a_1}^{f(a_1)} \cdots {a_d}^{f(a_d)}$ such that $(a_i)^{-m}$, $m > 0$, is represented by $({a_i}^-)^m$, $1 \leq i \leq d$. Besides this canonical representation of f by the string ${a_1}^{f(a_1)} \cdots {a_d}^{f(a_d)}$, any other string having the same Parikh vector with respect to the (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_d, {a_d}^- \rangle$ can be used for representing the multiset given by f as well. According to these definitions, matter and related anti-matter cannot be present in the same string or multiset over the alphabet $\{a_1, {a_1}^-, \cdots, a_d, {a_d}^-\}$. Obviously, their is a one-to-one correspondence between vectors from $\mathbb{Z}^d$ and the corresponding Parikh vectors over $\langle a_1, {a_1}^-, \cdots, a_d, {a_d}^- \rangle$, which can also be viewed as vectors over $\mathbb{Z}^{2d}$: for any of these vectors $v = (v_1, v_2, \cdots, v_{2d-1}, v_{2d})$, we have either $v_{2i-1} = 0$ or $v_{2i} = 0$ (or both), for all $1 \leq i \leq d$.

In order to specify that now we are dealing with d-dimensional vectors of integer numbers, we use the notation $Ps^{\mathbb{Z}^d}$: the family of sets of integer numbers $Ps_\delta^{\mathbb{Z}^d}(\Pi)$, $\delta \in \{gen, acc, aut\}$, and the family of functions/relations $ZPs_\alpha^{\mathbb{Z}^d}(\Pi)$, $Z \in \{Fun, Rel\}$, $\alpha \in \{acc, aut\}$, computed by such P systems with at most m membranes and k catalysts is denoted by $Ps_\delta^{\mathbb{Z}^d} OP_m(cat(k), antim/pri)$ and $ZPs_\alpha^{\mathbb{Z}^d} OP_m(cat(k), antim/pri)$; we omit $/pri$ for the families without priorities. Moreover, the family of recursively enumerable sets of d-dimensional vectors of integer numbers is denoted by $Ps^{\mathbb{Z}^d} RE$, the corresponding functions/relations by $ZPs^{\mathbb{Z}^d} RE$.

Theorem 7. *For any $d \geq 1$ we have that:*

- *for any $n \geq 1$, $k \geq 0$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$Ps_\delta^{\mathbb{Z}^d} OP_n(cat(k), antim/pri) = Ps^{\mathbb{Z}^d} RE \text{ and}$$
$$ZPs_\alpha^{\mathbb{Z}^d} OP_n(cat(k), antim/pri) = ZPs^{\mathbb{Z}^d} RE;$$

- *for any* $n \geq 1$, $k \geq 1$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, *and* $Z \in \{Fun, Rel\}$,

$$Ps_{\delta}^{\mathbb{Z}^d} OP_n\,(cat(k), antim) = Ps^{\mathbb{Z}^d} RE \text{ and}$$
$$ZPs_{\alpha}^{\mathbb{Z}^d} OP_n\,(cat(k), antim) = ZPs^{\mathbb{Z}^d} RE;$$

- *for any* $n \geq 1$, *and* $k \geq 0$,

$$Ps_{detacc}^{\mathbb{Z}^d} OP_n\,(cat(k), antim/pri) = Ps^{\mathbb{Z}^d} RE \text{ and}$$
$$FunPs_{detacc}^{\mathbb{Z}^d} OP_n\,(cat(k), antim/pri) = FunZPs^{\mathbb{Z}^d} RE.$$

Proof. As we have shown in Section 4, all variants of P systems with anti-matter mentioned in the theorem are computationally complete when dealing with multisets over any arbitrary alphabet, being able to simulate the actions of a register machine. Hence, as any d-dimensional vector of integer numbers can be represented by a $2d$-dimensional vector of non-negative integers, which can be processed in the usual way by register machines and thus simulated by all the variants of P systems with anti-matter mentioned in the theorem, we only have to solve the technical detail how to get this $2d$-dimensional vector of non-negative integers from a given d-dimensional vector of integer numbers represented by symbols over the (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_d, {a_d}^- \rangle$: given the input in an input membrane $\neq 0$, we there just make a first step using in parallel the non-cooperative rules $a_i \to [a_i, +]$ and ${a_i}^- \to [a_i, -]$, $1 \leq i \leq d$. Then the multisets over these symbols can be handled in the usual way, now both of them having the corresponding anti-matter objects $[a_i, +]^-$ and $[a_i, -]^-$. In a similar way, we can take the input from the environment by using rules of the form $q \to p\,(a_i, come)\,[a_i, +]$ or $q \to p\,({a_i}^-, come)\,[a_i, -]$ where q, p represent states of the register machine. The symbols a_i and ${a_i}^-$ then are not needed any more and can be eliminated by the rules $a_i \to \lambda$ and ${a_i}^- \to \lambda$. The remaining computations in the respective P system then can be carried out by simulating the actions of a register machine. □

8 Computing with Languages

P systems with anti-matter, as most of the computationally complete variants of P systems, can also be considered as language generating devices – the objects sent out can be concatenated to strings over a given alphabet, and the objects taken in during a halting computation can be assumed to form a string. For sake of simplicity, we may assume that in each computation step, at most one symbol is sent out or taken in; otherwise, as usual, e.g., see [3], we may take any permutation of the symbols sent out or taken in to be part of a string to be considered as output or input, respectively. Obviously, according to this method of getting an input string, for the accepting case only the automaton variant is to be considered now, as otherwise we would have to take an encoding of the input string by a multiset.

8.1 Languages over Strings

Let V be a finite alphabet. The set of strings (over V) generated or accepted (in the sense of automata) by a P system with anti-matter Π is denoted by $L_\delta^V(\Pi)$, $\delta \in \{gen, aut\}$, the function/relation computed by Π is denoted by $ZL_{aut}^V(\Pi)$, $Z \in \{Fun, Rel\}$. The family of sets $L_\delta^V(\Pi)$, $\delta \in \{gen, aut\}$, and the family of functions/relations $ZL_{aut}^V(\Pi)$, $Z \in \{Fun, Rel\}$, computed by such P systems with at most m membranes and k catalysts is denoted by $L_\delta^V OP_m(cat(k), antim/pri)$ and $ZL_{aut}^V OP_m(cat(k), antim/pri)$, respectively; we omit $/pri$ for the families without priorities; $cat(0)$ is used as a synonym for $ncoo$. If the alphabet is arbitrary, we omit the superscript V in these notations. Moreover, the languages over V in RE are denoted by RE^V, the corresponding family of functions/relations by ZRE^V.

The use of anti-matter and of matter/anti-matter annihilation rules (having priority over other rules) allows us to give a simple example how to generate an even non-context-free string language:

Example 1. Consider the P system with anti-matter

$$\begin{aligned}
\Pi &= (O, [\ ]_1, q_1, R_1, 1) \text{ where} \\
O &= \{a, b, c\} \cup \{b^-, c^-\} \cup \{q_1, q_2, q_3\}, \\
R_1 &= \{q_1 \to q_2, q_2 \to q_3, q_3 \to \lambda, q_1 \to q_1(a, come)\, b^- c^-\} \\
&\quad \cup \{q_2 \to q_2(b, come), q_3 \to q_3(c, come)\} \\
&\quad \cup \{a \to \lambda\} \cup \{x \to x, x^- \to x^-, xx^- \to \lambda \mid x \in \{b, c\}\}.
\end{aligned}$$

The reader may easily verify that

$$L_{aut}^{\{a,b,c\}}(\Pi) = \{a^n b^n c^n \mid n \geq 0\}.$$

For each symbol a taken in with state q_1 (which is eliminated in the next step by $a \to \lambda$) using the rule $q_1 \to q_1(a, come)\, b^- c^-$, an anti-matter object for both b and c is generated. The anti-matter objects b^- are eliminated in state q_2, and afterwards the anti-matter objects c^- are eliminated in state q_3. The computation only halts (with empty skin membrane) after having used the rule $q_3 \to \lambda$ if and only if an equal number of objects a, b, and c has been taken in, as otherwise, the rules $x \to x$ or $x^- \to x^-$, $x \in \{b, c\}$, keep the system in an infinite loop if too many x or not enough x have been taken in, respectively. Observe that this system also works if we do not require priority of the annihilation rules, but then, for each successful computation accepting the string $a^n b^n c^n$, $n \geq 1$, there exist inifinite computations where we use one of the rules $x^- \to x^-$ again and again instead of letting x^- being annihilated by $xx^- \to \lambda$. Hence, we may say that

$$\{a^n b^n c^n \mid n \geq 0\} \in L_{aut}^{\{a,b,c\}} OP_1(ncoo).$$

Theorem 8. *For any arbitrary alphabet V we have that:*

- *for any* $n \geq 1$, $k \geq 0$, $\delta \in \{gen, aut\}$, *and* $Z \in \{Fun, Rel\}$,

$$L_{\delta}^{V} OP_n\,(cat(k), antim/pri) = RE^V \text{ and}$$
$$ZL_{aut}^{V} OP_n\,(cat(k), antim/pri) = ZRE^V;$$

- *for any* $n \geq 1$, $k \geq 1$, $\delta \in \{gen, aut\}$, *and* $Z \in \{Fun, Rel\}$,

$$L_{\delta}^{V} OP_n\,(cat(k), antim) = RE^V \quad \text{and}$$
$$ZL_{aut}^{V} OP_n\,(cat(k), antim) = ZRE^V.$$

Proof. As we have shown in Section 4, all variants of P systems with anti-matter mentioned in the theorem are computationally complete when dealing with multisets, being able to simulate the actions of a register machine. Hence, by well-known techniques, input symbols composing an input string can be encoded as numbers in an input register and thus as a multiset in the simulating P system with anti-matter. In the same way, the results of a computation in the P system can be decoded from the multiset representing the output register of the underlying register machine. An input symbol $a \in V$ is taken in by rules of the form $q \to p\,(a, come)$ where q, p represent states of the register machine, and sent out by rules of the form $q \to p\,(a, out)$. □

8.2 Languages over Computable Finite Presentations of Groups

Strings may be used in a wider sense as repesentations of group elements. In order to establish these more general results, we first need some definitions and examples from group theory, e.g., see [11].

Groups and Group Presentations

Let $G = (G', \circ)$ be a group with group operation $\circ$. As is well-known, the group axioms are

- *closure*: for any $a, b \in G'$, $a \circ b \in G'$,
- *associativity*: for any $a, b, c \in G'$, $(a \circ b) \circ c = a \circ (b \circ c)$,
- *identity*: there exists a (unique) element $e \in G'$, called the *identity*, such that $e \circ a = a \circ e$ for all $a \in G'$, and
- *invertibility:* for any $a \in G'$, there exists a (unique) element a^{-1}, called the *inverse* of a, such that $a \circ a^{-1} = a^{-1} \circ a = e$.

Moreover, the group is called *commutative*, if for any $a, b \in G'$, $a \circ b = b \circ a$. In the following, we will not distinguish between G' and G if the group operation is obvious from the context.

For any element $b \in G'$, the order of b is the smallest number $n \in \mathbb{N}$ such that $b^n = e$ provided such an n exists, and then we write $ord\,(b) = n$; if no such n exists, $\{b^n \mid n \geq 1\}$ is an infinite subset of G' and we write $ord\,(b) = \infty$.

For any set B, B^{-1} is defined as the set of symbols representing the inverses of the elements of B, i.e., $B^{-1} = \left\{b^{-1} \mid b \in B\right\}$. We now consider the strings in $\left(B \cup B^{-1}\right)^*$ and two strings as different unless their equality follows from the group axioms, i.e., for any $a, b, c \in \left(B \cup B^{-1}\right)^*$, $a \circ b \circ b^{-1} \circ c = a \circ c$; using these reductions, we obtain a set of irreducible strings from those in $\left(B \cup B^{-1}\right)^*$, the set of which we denote by $I(B)$. Then the *free group* generated by B is $F(B) = (I(B), \circ)$ with the elements being the irreducible strings over $B \cup B^{-1}$ and the group operation to be interpreted as the usual string concatenation, yet, obviously, if we concatenate two elements from $I(B)$, the resulting string eventually has to be reduced again. The identity in $F(B)$ is the empty string.

In general, B (not containing the identity) is called a *generator* of the group G if every element a from G can be written as a finite product/sum of elements from B, i.e., $a = b_1 \circ \cdots \circ b_m$ for $b_1, \ldots, b_m \in B$. In this paper, we restrict ourselves to finitely presented groups, i.e., having a finite presentation $\langle B \mid R \rangle$ with B being a finite generator set and moreover, R being a finite set of relations among these generators. In a similar way as in the definition of the free group generated by B, we here consider the strings in B^* reduced according to the group axioms and the relations given in R. Informally, the group $G = \langle B \mid R \rangle$ is the largest one generated by B subject only to the group axioms and the relations in R. Formally, we will restrict ourselves to relations of the form $b_1 \circ \cdots \circ b_m = c^{-1}$ with $b_1, \ldots, b_m, c \in B$, which equivalently may be written as $b_1 \circ \cdots \circ b_m \circ c = e$; hence, instead of such relations we may specify R by strings over B yielding the group identity, i.e., instead of $b_1 \circ \cdots \circ b_m = c^{-1}$ we take $b_1 \circ \cdots \circ b_m \circ c$ (these strings then are called *relators*).

Example 2. The free group $F(B) = (I(B), \circ)$ can be written as $\langle B \mid \emptyset \rangle$ (or even simpler as $\langle B \rangle$) because it has no restricting relations.

Example 3. The *cyclic group* of order n has the presentation $\langle \{a\} \mid \{a^n\} \rangle$ (or, omitting the set brackets, written as $\langle a \mid a^n \rangle$); it is also known as $\mathbb{Z}_n$ or as the quotient group $\mathbb{Z}/\mathbb{Z}_n$.

Example 4. $\mathbb{Z}$ is a special case of an Abelian group generated by (1) and its inverse (-1), i.e., $\mathbb{Z}$ is the free group generated by (1). $\mathbb{Z}^d$ is an Abelian group generated by the unit vectors $(0, ..., 1, ..., 0)$ and their inverses $(0, ..., -1, ..., 0)$. It is well known that every finitely generated Abelian group is a direct sum of a torsion group and a free Abelian group where the torsion group may be written as a direct sum of finitely many groups of the form $\mathbb{Z}/p^k\mathbb{Z}$ for p being a prime, and the free Abelian group is a direct sum of finitely many copies of $\mathbb{Z}$.

Example 5. A very well-known example for a non-Abelian group is the hexagonal group with the finite presentation $\left\langle a, b, c \mid a^2, b^2, c^2 \right\rangle$. All three generators a, b, c are self-inverse.

Remark 3. Unfortunately, given a finite presentation of a group $\langle B \mid R \rangle$, in general it is not even decidable whether the group presented in that way is finite or

infinite. Hence, in this paper we restrict ourselves to infinite groups where the word equivalence problem $u = v$ is decidable, or equivalently, there is a decision procedure telling us whether, given two strings u and v, $u \circ v^{-1} = e$. In that case, we call $\langle B \mid R \rangle$ a *recursive* or *computable* finite group presentation.

As a first example we now consider the set ("language") of all one-dimensional vectors:

Example 6. Consider the P system

$$\begin{aligned} \Pi &= (\{q_0, q_+, q_-, q_h\}, [\ ]_1, q_0, R_1, 1) \text{ where} \\ R_1 &= \{q_0 \to q_h, q_+ \to q_h, q_- \to q_h\} \\ &\quad \cup \{q_0 \to (+1)q_+, q_+ \to (+1)q_+, q_0 \to (-1)q_-, q_- \to (-1)q_-\}. \end{aligned}$$

In order to generate the empty string, corresponding with the zero-vector (0), we simply apply $q_0 \to q_h$. We may also choose to generate a positive or a negative vector, i.e., we start with $q_0 \to (+1)q_+$ or $q_0 \to (-1)q_-$, respectively. After $n-1$ applications of the rules $q_+ \to (+1)q_+$ and $q_- \to (-1)q_-$ as well as of the final rule $q_+ \to q_h$ or $q_- \to q_h$, respectively, we have sent out a string representing the unique irreducible representation of the vector $(+n)$ or $(-n)$, respectively.

Remark 4. The reader may easily verify that, given any finitely generated Abelian group, such a regular P system exists which generates all strings representing the (unique, with respect to a complete order on the positive generators) irreducible representations of the group elements. For non-commutative groups with relators, such trivial representations are not possible.

If we do not require irreducibility of the string sent out to the environment, then of course, for any finitely generated group, we can generate representations of all its elements very easily:

Example 7. Given a finite presentation of a group $\langle B \mid R \rangle$, with $B^- = B$, consider the P system

$$\begin{aligned} \Pi &= (\{q_0\}, [\ ]_1, q_0, R_1, 1) \text{ where} \\ R_1 &= \{q_0 \to \lambda\} \cup \{q_0 \to g q_0 \mid g \in B\}. \end{aligned}$$

Most of the strings sent out now will not be reduced.

Remark 5. In general, as long as we have given the group by a computable finite presentation, for a mechanism having the full power of Turing computability, we can require that the "strings" sent out to the environment are irreducible ones. Hence, for a given recursively enumerable set L of elements over the computable finite presentation $\langle B \mid R \rangle$ of a group, such a mechanism can generate the irreducible string representations of the elements in L. Thus, the results collected in the following theorem are obvious consequences of the results stated in Theorem 8.

Let $\langle B \mid R \rangle$ be the computable finite presentation of a group. The set of string representations (of elements of this group with respect to this finite presentation $\langle B \mid R \rangle$) generated or accepted (in the sense of automata) by a P system with anti-matter Π is denoted by $L_{\delta}^{\langle B|R \rangle}(\Pi)$, $\delta \in \{gen, aut\}$, the function/relation computed by Π is denoted by $ZL_{aut}^{\langle B|R \rangle}(\Pi)$, $Z \in \{Fun, Rel\}$. The family of sets $L_{\delta}^{\langle B|R \rangle}(\Pi)$, $\delta \in \{gen, aut\}$, and the family of functions/relations $ZL_{aut}^{\langle B|R \rangle}(\Pi)$, $Z \in \{Fun, Rel\}$, computed by such P systems with at most m membranes and k catalysts is denoted by $L_{\delta}^{\langle B|R \rangle}OP_m(cat(k), antim/pri)$ and $ZL_{aut}^{\langle B|R \rangle}OP_m(cat(k), antim/pri)$, respectively; we omit $/pri$ for the families without priorities. If the computable finite group presentation may be an arbitrary one, we omit the superscript $\langle B \mid R \rangle$ in these notations. The family of recursively enumerable sets of elements over the computable finite presentation $\langle B \mid R \rangle$ of a group is denoted by $RE^{\langle B|R \rangle}$, the corresponding family of recursively enumerable functions/relations by $ZRE^{\langle B|R \rangle}$, $Z \in \{Fun, Rel\}$.

Theorem 9. *Let $\langle B \mid R \rangle$ be the computable finite presentation of a group. Then we have that:*

- *for any $n \geq 1$, $k \geq 0$, $\delta \in \{gen, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$L_{\delta}^{\langle B|R \rangle}OP_n(cat(k), antim/pri) = RE^{\langle B|R \rangle} \text{ and}$$
$$ZL_{aut}^{\langle B|R \rangle}OP_n(cat(k), antim/pri) = ZRE^{\langle B|R \rangle};$$

- *for any $n \geq 1$, $k \geq 1$, $\delta \in \{gen, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$L_{\delta}^{\langle B|R \rangle}OP_n(cat(k), antim) = RE^{\langle B|R \rangle} \quad \text{and}$$
$$ZL_{aut}^{\langle B|R \rangle}OP_n(cat(k), antim) = ZRE^{\langle B|R \rangle}.$$

Proof. As for string languages, all computations can be carried out by simulating register machines, hence, again the results from Section 4 apply. Moreover, as already mentioned in Remark 5, the additional computations can also be carried out in this way, as $\langle B \mid R \rangle$ is computable. □

Remark 6. Let us mention that the results obtained in Theorem 9 for arbitrary computable finite presentations $\langle B \mid R \rangle$ of a group can also be applied to the infinite Abelian groups $\mathbb{Z}^d$ with their canonical group presentations by the unit vectors $(0, ..., 1, ..., 0)$ and their inverses $(0, ..., -1, ..., 0)$. Keeping in mind that there is a one-to-one correspondence between the representation of a vector in $\mathbb{Z}^n$ by a multiset of symbols and the corresponding string representing this multiset, most of the results shown in Theorem 7 are special cases of the respective results stated in Theorem 9.

9 Summary

We have shown that only non-cooperative rules together with matter/anti-matter annihilation rules are needed to obtain computational completeness in P systems

working in the maximally parallel derivation mode if annihilation rules have weak priority; without priorities, one catalyst is needed. In the case of accepting P systems we were able to even get deterministic systems. Allowing anti-matter objects as input and/or output, we have even obtained a computationally complete computing model for computations on integer numbers. Interpreting sequences of symbols taken in from and/or sent out to the environment, we have also got a model for computations on strings, where strings can even be interpreted as representations of elements of a group based on a computable finite presentation.

There may be a lot of other interesting models of P systems allowing for introducing anti-matter objects and matter/anti-matter annihilation rules. Several problems remain open even for the models presented here, for example, can we avoid both catalysts and priorities. Moreover, the number of rules needed for universal P systems with anti-matter might still be reduced. Finally, the variants of P systems with anti-matter computing on sets of integer numbers and on languages of strings, even considered as representations of elements of a group based on a computable finite presentation, deserve more detailed investigations.

References

1. A. Alhazov, D. Sburlan: Static Sorting P Systems. In: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.): Applications of Membrane Computing. *Natural Computing Series*, Springer, 2005, pp. 215–252.
2. M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-Related Outputs of Computations in P Systems. *Journal of Automata, Languages and Combinatorics* **11** (3), 263–278 (2006).
3. E. Csuhaj-Varjú, Gy. Vaszil: P Automata or Purely Communicating Accepting P Systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing. International Workshop, WMC-CdeA 2002 Curtea de Argeş, Romania, August 19–23, 2002. Revised Papers*. Lecture Notes in Computer Science **2597**, Springer, 2003, pp. 219–233.
4. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
5. D. Díaz-Pernil, F. Peña-Cantillana, M. A. Gutiérrez-Naranjo: Antimatter as a Frontier of Tractability in Membrane Computing. *Brainstorming Week in Membrane Computing*, Sevilla, February 2014.
6. R. Freund: Purely Catalytic P Systems: Two Catalysts Can Be Sufficient for Computational Completeness. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin (Eds.): *CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013*. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 2013, pp. 153–166.
7. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science* **330**, 251–266 (2005).
8. R. Freund, M. Oswald: A Small Universal Antiport P System with Forbidden Context. In: H. Leung, G. Pighizzini (Eds.): *8th International Workshop on Descriptional Complexity of Formal Systems - DCFS 2006*, Las Cruces, New Mexico, USA, June

21 - 23, 2006. Proceedings DCFS, New Mexico State University, Las Cruces, New Mexico, USA, 2006, pp. 259–266.
9. R. Freund, M. Oswald: Catalytic and Purely Catalytic P Automata: Control Mechanisms for Obtaining Computational Completeness. In: S. Bensch, F. Drewes, R. Freund, F. Otto (Eds.): *Fifth Workshop on Non-Classical Models of Automata and Applications (NCMA 2013)*, OCG, Wien, 2013, pp. 133–150.
10. R. Freund, Gh. Păun: How to Obtain Computational Completeness in P Systems with One Catalyst. In: T. Neary and M. Cook: *Proceedings Machines, Computations and Universality 2013, MCU 2013*, Zürich, Switzerland, September 9–11, 2013, *EPTCS* **128**, 47–61 (2013).
11. D. F. Holt, B. Eick, E. A. O'Brien: *Handbook of Computational Group Theory.* CRC Press, 2005.
12. I. Korec: Small Universal Register Machines. *Theoretical Computer Science* **168**, 267–301 (1996).
13. M. L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
14. L. Pan, Gh. Păun: Spiking Neural P Systems with Anti-Matter. *International Journal of Computers, Communications & Control* **4** (3), 273–282 (2009).
15. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
16. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
17. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
18. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.
19. The P Systems Website: `www.ppage.psystems.eu`.

Priorities, Promoters and Inhibitors in Deterministic Non-Cooperative P Systems

Artiom Alhazov[1], Rudolf Freund[2]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
artiom@math.md
[2] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
rudi@emcc.at

Summary. Membrane systems (with symbol objects) are distributed controlled multiset processing systems. Non-cooperative P systems with either promoters or inhibitors (of weight not restricted to one) are known to be computationally complete. Since recently, it is known that the power of the deterministic subclass of such systems is subregular. We present new results on the weight of promoters and inhibitors, as well as for characterizing the systems with priorities only.

1 Introduction

The most famous membrane computing model where determinism is a criterion of universality versus decidability is the model of catalytic P systems, see [3] and [6].

It is also known that non-cooperative rewriting P systems with either promoters or inhibitors are computationally complete, [2]. Moreover, the proof satisfies some additional properties:

- Either promoters of weight 2 or inhibitors of weight 2 are enough.
- The system is non-deterministic, but it restores the previous configuration if the guess is wrong, which leads to correct simulations with probability 1.

Recently, in [1] it was shown that the computational completeness cannot be achieved by deterministic non-cooperative systems with promoters, inhibitors and priorities (in maximally parallel or asynchronous mode, unlike the sequential mode), and characterizations of the corresponding classes were obtained:

$$\begin{aligned} NFIN \cup coNFIN &= N_{deta}OP_1^{asyn}\left(ncoo, pro_{1,*}, inh_{1,*}\right) \\ &= N_{deta}OP_1^{maxpar}\left(ncoo, pro_{1,*}\right) \\ &= N_{deta}OP_1^{maxpar}\left(ncoo, inh_{1,*}\right) \\ &= N_{deta}OP_1^{asyn}\left(ncoo, \left(pro_{*,*}, inh_{*,*}\right)_*, pri\right) \\ &= N_{deta}OP_1^{maxpar}\left(ncoo, \left(pro_{*,*}, inh_{*,*}\right)_*, pri\right), \text{ but} \\ NRE &= N_{deta}OP_1^{sequ}\left(ncoo, pro_{1,1}, inh_{1,1}\right). \end{aligned}$$

A few interesting questions have been left open. For instance, what is the power of P systems, e.g., in the maximally parallel mode, when we only use priorities, or when we restrict the weight of the promoting/inhibiting multisets. These are the questions we address in this paper.

2 Definitions

An *alphabet* is a finite non-empty set V of abstract *symbols*. The free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. The set of non-negative integers is denoted by $\mathbb{N}$; a set S of non-negative integers is called *co-finite* if $\mathbb{N} \setminus S$ is finite. The family of all finite (co-finite) sets of non-negative integers is denoted by $NFIN$ ($coNFIN$, respectively). The family of all recursively enumerable sets of non-negative integers is denoted by NRE. In the following, we will use $\subseteq$ both for the subset as well as the submultiset relation.

Since flattening the membrane structure of a membrane system preserves both determinism and the model, in the following we restrict ourselves to consider membrane systems as one-region multiset rewriting systems.

A *(one-region) membrane system (P system)* is a tuple

$$\Pi = (O, \Sigma, w, R'),$$

where O is a finite alphabet, $\Sigma \subseteq O$ is the input sub-alphabet, $w \in O^*$ is a string representing the initial multiset, and R' is a set of rules of the form $r : u \to v$, $u \in O^+$, $v \in O^*$.

A configuration of the system Π is represented by a multiset of objects from O contained in the region, the set of all configurations over O is denoted by $\mathbb{C}(O)$. A rule $r : u \to v$ is applicable if the current configuration contains the multiset specified by u. Furthermore, applicability may be controlled by *context conditions*, specified by pairs of sets of multisets.

Definition 1. *Let P_i, Q_i be (finite) sets of multisets over O, $1 \leq i \leq m$. A* rule with context conditions $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$ *is applicable to a configuration C if r is applicable, and there exists some $j \in \{1, \cdots, m\}$ for which*

- *there exists some $p \in P_j$ such that $p \subseteq C$ and*
- *$q \not\subseteq C$ for all $q \in Q_j$.*

In words, context conditions are satisfied if there exists a pair of sets of multisets (called *promoter set* and *inhibitor set*, respectively) such that at least one multiset in the promoter set is a submultiset of the current configuration, and no multiset in the inhibitor set is a submultiset of the current configuration.

Definition 2. *A* P system with context conditions and priorities on the rules *is a construct*

$$\Pi = (O, \Sigma, w, R', R, >),$$

where (O, Σ, w, R') *is a (one-region) P system as defined above,* R *is a set of rules with context conditions and* $>$ *is a priority relation on the rules in* R*; if rule* r' *has priority over rule* r*, denoted by* $r' > r$*, then* r *cannot be applied if* r' *is applicable.*

Throughout the paper, we will use the word *control* to mean that at least one of these features is allowed (context conditions or promoters or inhibitors only and eventually priorities).

In the *sequential mode* (*sequ*), a computation step consists in the non-deterministic application of one applicable rule r, replacing its left-hand side ($lhs\,(r)$) with its right-hand side ($rhs\,(r)$). In the *maximally parallel mode* (*maxpar*), a multiset of applicable rules may be chosen non-deterministically to be applied in parallel to the underlying configuration to disjoint submultisets, possibly leaving some objects idle, under the condition that no further applicable rule can be added to that multiset (i.e., no supermultiset of the chosen multiset is applicable to the same configuration). Maximal parallelism is the most common computation mode in membrane computing, see also Definition 4.8 in [5]. In the *asynchronuous mode* (*asyn*), any positive number of applicable rules may be chosen non-deterministically to be applied in parallel to the underlying configuration, to disjoint submultisets. The computation step between two configurations C and C' is denoted by $C \to C'$, thus yielding the binary relation $\Rightarrow: \mathbb{C}(O) \times \mathbb{C}(O)$. A computation halts when there are no rules applicable to the current configuration (*halting configuration*) in the corresponding mode.

The computation of a *generating* P system starts with w, and its result is $|x|$ if it halts, an *accepting* system starts with wx, $x \in \Sigma^*$, and we say that $|x|$ is its results – is accepted – if it halts. The set of numbers generated/accepted by a P system working in the mode α is the set of results of its computations for all $x \in \Sigma^*$ and denoted by $N_g^\alpha(\Pi)$ and $N_a^\alpha(\Pi)$, respectively. The family of sets of numbers generated/accepted by a family of (one-region) P systems with context conditions and priorities on the rules with rules of type β working in the mode α is denoted by $N_\delta OP_1^\alpha\left(\beta, (pro_{k,l}, inh_{k',l'})_d, pri\right)$ with $\delta = g$ for the generating and $\delta = a$ for the accepting case; d denotes the maximal number m in the rules with context conditions $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$; k and k' denote the maximum number of promoters/inhibitors in the P_i and Q_i, respectively; l and l' indicate the maximum of weights of promotors and inhibitors, respectively. If any of these numbers k, k', l, l' is not bounded, we replace it by $*$. As types of rules we are going to distinguish between cooperative ($\beta = coo$) and non-cooperative (i.e., the left-hand side of each rule is a single object; $\beta = ncoo$) ones.

In the case of accepting systems, we also consider the idea of determinism, which means that in each step of any computation at most one (multiset of) rule(s) is applicable; in this case, we write *deta* for δ.

In the literature, we find a lot of restricted variants of P systems with context conditions and priorities on the rules, e.g., we may omit the priorities or the context conditions completely. If in a rule $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$ we have $m = 1$, we say that $(r, (P_1, Q_1))$ is a rule with a *simple context condition*, and we omit the inner parentheses in the notation. Moreover, context conditions only using promoters are denoted by $r|_{p_1,\cdots,p_n}$, meaning $(r, \{p_1, \cdots, p_n\}, \emptyset)$, or, equivalently, $(r, (p_1, \emptyset), \cdots, (p_n, \emptyset))$; context conditions only using inhibitors are denoted by $r|_{\neg q_1,\cdots,\neg q_n}$, meaning $(r, \lambda, \{q_1, \cdots, q_n\})$, or $r|_{\neg\{q_1,\cdots,q_n\}}$. Likewise, a rule with both promoters and inhibitors can be specified as a rule with a simple context condition, i.e., $r|_{p_1,\cdots,p_n,\neg q_1,\cdots,\neg q_n}$ stands for $(r, \{p_1, \cdots, p_n\}, \{q_1, \cdots, q_n\})$. Finally, promoters and inhibitors of weight one are called *atomic*.

Remark 1. If we do not consider determinism, then (the effect of) the rule $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$ is equivalent to (the effect of) the collection of rules $\{(r, P_j, Q_j) \mid 1 \leq j \leq m\}$, no matter in which mode the P system is working (obviously, the priority relation has to be adapted accordingly, too).

Remark 2. Let $(r, \{p_1, \cdots, p_n\}, Q)$ be a rule with a simple context condition; then we claim that (the effect of) this rule is equivalent to (the effect of) the collection of rules

$$\{(r, \{p_j\}, Q \cup \{p_k \mid 1 \leq k < j\}) \mid 1 \leq j \leq m\}$$

even in the the case of a deterministic P system: If the first promoter is chosen to make the rule r applicable, we do not care about the other promoters; if the second promoter is chosen to make the rule r applicable, we do not allow p_1 to appear in the configuration, but do not care about the other promoters p_3 to p_m; in general, when promoter p_j is chosen to make the rule r applicable, we do not allow p_1 to p_{j-1} to appear in the configuration, but do not care about the other promoters p_{j+1} to p_m; finally, we have the rule $\{(r, \{p_m\}, Q \cup \{p_k \mid 1 \leq k < m\})\}$. If adding $\{p_k \mid 1 \leq k < j\}$ to Q has the effect of prohibiting the promotor p_j from enabling the rule r to be applied, this makes no harm as in this case one of the promoters p_k, $1 \leq k < j$, must have the possibility for enabling r to be applied. By construction, the domains of the new context conditions now are disjoint, so this transformation does not create (new) non-determinism. In a similar way, this transformation may be performed on context conditions which are not simple. Therefore, without restricting generality, the set of promoters may be assumed to be a singleton. In this case, we may omit the braces of the multiset notation for the promoter multiset and write (r, p, Q).

Remark 3. As in a P system $(O, \Sigma, w, R', R, >)$ the set of rules R' can easily be deduced from the set of rules with context conditions R, we omit R' in the description of the P system. Moreover, for systems having only rules with a simple

context condition, we omit d in the description of the families of sets of numbers and simply write

$$N_\delta OP_1^\alpha\left(\beta, pro_{k,l}, inh_{k',l'}, pri\right).$$

Moreover, each control mechanism not used can be omitted, e.g., if no priorities and only promoters are used, we only write $N_\delta OP_1^\alpha\left(\beta, pro_{k,l}\right)$.

3 Results

3.1 Recent results

We first recall from [1] the *bounding* operation over multisets, with a parameter $k \in \mathbb{N}$ as follows:

$$\text{for } u \in O^*,\ b_k(u) = v \text{ with } |v|_a = \min(|u|_a, k) \text{ for all } a \in O.$$

The mapping b_k "crops" the multisets by removing copies of every object a present in more than k copies until exactly k remain. For two multisets u, u', $b_k(u) = b_k(u')$ if for every $a \in O$, either $|u|_a = |u'|_a < k$, or $|u|_a \geq k$ and $|u'|_a \geq k$. Mapping b_k induces an equivalence relation, mapping O^* into $(k+1)^{|O|}$ equivalence classes. Each equivalence class corresponds to specifying, for each $a \in O^*$, whether no copy, one copy, or $\cdots\ k-1$ copies, or "k copies or more" are present. We denote the range of b_k by $\{0, \cdots, k\}^O$.

Lemma 1. *[1] Context conditions are equivalent to predicates defined on boundings.*

Theorem 1. *[1] Priorities are subsumed by conditional contexts.*

Remark 4. It is worth to note, see also [4], that if no other control is used, the priorities can be mapped to sets of atomic inhibitors. Indeed, a rule is inhibited precisely by the left side of each higher priority rule. This is straightforward in case when the priority relation is assumed to be a partial order.

If it is not, then both the semantics of computation in P systems and the reduction of priorities to inhibitors is a bit more complicated, but the claim still holds.

Fix an arbitrary deterministic controlled non-cooperative P system. Take k as the maximum of size of all multisets in all context conditions. Then, the bounding does not influence applicability of rules, and $b_k(u)$ is halting if and only if u is halting. We recall that bounding induces equivalence classes preserved by any computation.

Lemma 2. *[1] Assume $u \to x$ and $v \to y$. Then $b_k(u) = b_k(v)$ implies $b_k(x) = b_k(y)$.*

Corollary 1. *[1] If $b_k(u) = b_k(v)$, then u is accepted if and only if v is accepted.*

Finally, the "at most $NFIN \cup coNFIN$" part of characterizing

$$N_{deta}OP_1^{maxpar}\left(ncoo, (pro_{*,*}, inh_{*,*})_*, pri\right)$$

(the main theorem of [1]) is shown with the following argument:

> Each equivalence class induced by bounding is completely accepted or completely rejected. If no infinite equivalence class is accepted, then the accepted set is finite (containing numbers not exceeding $(k-1)\cdot|O|$). If at least one infinite equivalence class is accepted, then the rejected set is finite (containing numbers not exceeding $(k-1)\cdot|O|$).

3.2 Priorities only

We start with an example how to deterministically rewrite an object t depending on the presence or absence of object a.

Example 1.

$$\Pi = (\{a, A, A', t, t', t_+, t_-\}, \{a\}, tA, R, R, >), \text{ where}$$
$$R = \{1 : t \to t',\ 2 : a \to \lambda,\ 3 : A \to A',\ 4 : t' \to t_+,\ 5 : t' \to t_-,\ 6 : A' \to \lambda\},$$
$$> = \{a \to \lambda > A \to A',\ A \to A' > t' \to t_-,\ A' \to \lambda > t' \to t_+\}.$$

Indeed, object t waits for one step by becoming t', while A has to change to A' or wait, depending on the presence of a. Then, object t' becomes either t_+ or t_-, depending on whether A or A' is present. Notice, e.g., how adding either rule $t_+ \to t_+$ or rule $t_- \to t_-$ leads to a system accepting $\{0\}$ or $\mathbb{N} \setminus \{0\}$. Of course, accepting only zero could instead be done by a trivial one-rule system, but this example is important because such a deciding subsystem can be used, with suitable delays, as a building block for checking combinations of presence/absence of multiple symbols.

We now proceed with characterizing systems with priorities only.

Theorem 2. $N_{deta}OP_1^{maxpar}(ncoo, pri) = \{\mathbb{N}_k, \mathbb{N}_k \cup \{0\} \mid k \geq 0\} \cup \{\{0\}, \emptyset\}$.

Proof. We already know that the priorities correspond to sets of atomic inhibitors. This means that each system accepts a union of some equivalence classes induced by bounding b_1 (i.e., checking presence/absence). Note that various combinations of "$= 0$" and "≥ 1" yield numeric sets $\{0\}$ and $\mathbb{N}_k$ (where $k > 0$ is the number of different symbols present). The family of all unions of these sets is

$$F_{pri} = \{\mathbb{N}_k, \mathbb{N}_k \cup \{0\} \mid k \geq 0\} \cup \{\{0\}, \emptyset\}.$$

It follows that $N_{deta}OP_1^{maxpar}(ncoo, pri) \subseteq F_{pri}$.

We proceed with the converse inclusion. Let $\Pi_0 = (\{a, t\}, \{a\}, t, R, R, >)$, then $R = \{t \to t\}$ and empty relation $>$ yields $\emptyset$. To accept $\{0\}$, we instead take $R = \{a \to a\}$ and empty relation $>$.

Now suppose we want to accept $\mathbb{N}_k$. It would suffice to count that we have at least one of each objects $a_1, \cdots, a_k$ (we recall that we need to accept *at least one* input of size j for each $j \geq k$, or reject the input if $j > k$). To accept $\mathbb{N}_k \cup \{0\}$ instead, we may first perform a simultaneous check for the absence of all input symbols.

Using the idea from Example 1, we construct the system

$$
\begin{aligned}
\Pi_1 &= (O, \Sigma = \{a_{i,0} \mid 1 \leq i \leq k\}, tA_{0,0} \cdots A_{k,0}, R, R, >), \text{ where} \\
O &= \{a_{i,j} \mid 1 \leq i \leq k,\ 0 \leq j \leq i+1\} \cup \{A_{i,j} \mid 0 \leq i \leq k,\ 0 \leq j \leq i+2\} \\
&\quad \cup \{t, z, p\} \cup \{t_i \mid 0 \leq i \leq i+1\}, \\
R &= \{1 : a_{i,j} \to a_{i,j+1} \mid 1 \leq i \leq k,\ 0 \leq j \leq i\} \\
&\quad \cup \{2 : A_{i,j} \to A_{i,j+1} \mid 1 \leq i \leq k,\ 0 \leq j \leq i+1\} \\
&\quad \cup \{3 : t \to t_0,\ 4 : t_0 \to z,\ 5 : t_0 \to t_1,\ 6 : p \to p\} \\
&\quad \cup \{7 : t_i \to t_{i+1},\ 8 : t_i \to p \mid 1 \leq i \leq k\}, \\
> &= \{a_{i,0} \to a_{i,1} > A_{0,0} \to A_{0,1} \mid 1 \leq i \leq k\} \\
&\quad \cup \{A_{0,0} \to A_{0,1} > t_0 \to z,\ A_{0,1} \to A_{0,2} > t_0 \to t_1\} \\
&\quad \cup \{a_{i,i} \to a_{i,i+1} > A_{i,i} \to A_{i,i+1} \mid 1 \leq i \leq k\} \\
&\quad \cup \{A_{i,i} \to A_{i,i+1} > t_i \to p,\ A_{i,i+1} \to A_{i,i+2} > t_i \to t_{i+1}\}.
\end{aligned}
$$

Such system accept exactly $\mathbb{N}_k \cup \{0\}$. Indeed, after first step, $A_{0,0}$ is present if all input symbols were absent, otherwise $A_{0,1}$ is present instead. For any i, $1 \leq i \leq k$, after step $1+i$, object $A_{i,i}$ is present if input symbol $a_{i,0}$ was present in the input, and otherwise $A_{i,i+1}$ is present instead. These "decision symbols" are used by t_i, $0 \leq i \leq k$, to build the "presence picture". We recall that it suffices to accept when all input symbols are present, or when none of them is present. In the first case, t_0 becomes z, and the computation only continues by rules from groups 1 and 2, leading to halting. Let us assume that the first s of the input symbols are present, $s < k$. Then, t_0 becomes t_1, and then $\cdots$, t_s, and then the absence of t_{s+1} will change t_s into p, leading to an infinite computation. Finally, if all input symbols are present, then the computation will halt with t_{k+1}.

It remains to notice that accepting $\mathbb{N}_k$, $k \geq 1$, can be done by simply adding a rule $z \to z$. □

3.3 Promoters or inhibitors of weight 2

We start from examples, illustrating deterministic choice of rewriting p, depending on whether object a is absent, occurs exactly once, or occurs multiple times.

Example 2. Symbols A, B are primed if input is present (multiple input symbols are present). Then primed and unprimed symbols form mutually exclusive conditions.

$$\begin{aligned}
&\Pi = (O = \{p, p', p'', p_>, p_1, p_0, A, B, a\}, \Sigma = \{a\}, pAB, R', R), \text{ where}\\
&R' = \{1 : p \to p',\ 2 : A \to A',\ 3 : B \to B',\\
&\qquad 4 : p' \to p_>,\ 5 : p' \to p'',\ 6 : p'' \to p_1,\ 7 : p'' \to p_0\},\\
&R' = \{1 : p \to p',\ 2 : A \to A'|_a,\ 3 : B \to B'|_{aa},\\
&\qquad 4 : p' \to p_>|_B,\ 5 : p' \to p''|_{B'},\ 6 : p'' \to p_1|_A,\ 7 : p'' \to p_0|_{A'}\}.
\end{aligned}$$

Example 3. Notice that if we replace all promoters by inhibitors with the **same** context, the effect of blocking rules will be reversed, but the result will be the same. Indeed, the role of A' and B' will switch from found a and found aa, respectively, to not found a and not found aa, respectively.

$$\begin{aligned}
R' = \{&1 : p \to p',\ 2 : A \to A'|_{\neg a},\ 3 : B \to B'|_{\neg aa},\\
&4 : p' \to p_>|_{\neg B},\ 5 : p' \to p''|_{\neg B'},\ 6 : p'' \to p_1|_{\neg A},\ 7 : p'' \to p_0|_{\neg A'}\}.
\end{aligned}$$

We now proceed with characterizing systems with context of weight two. Notice that we already know that their power does not exceed $NFIN \cup coNFIN$.

Theorem 3. $N_{deta}OP_1^{maxpar}(ncoo, pro_2) =$
$N_{deta}OP_1^{maxpar}(ncoo, inh_2) = NFIN \cup coNFIN.$

Proof. We use the technique from Example 2 for all input symbols and combine the extracted information. Consider an arbitrary finite set M, and let $\max(M) = n$. We will use the following strategy: to accept a number $j \in M$, we will accept an input multiset with exactly j symbols appearing once, and nothing else. To accept the complement of M, we split it into sets $M'' = \{j \mid j > n\}$ and $M' = \{j \mid j \leq n,\ j \notin M\}$. While M' is treated similarly to M, it only remains to accept M'', which is covered by equivalence classes when all symbols are present, and at least one is present more than once.

$$\begin{aligned}
&\Pi = (O, \Sigma = \{a_i \mid 1 \leq i \leq n\}, tA_1 \cdots A_n B_1 \cdots B_n, R', R), \text{ where}\\
&O = \{t_{i,j}, T_{i,j}, t'_{i,j}, T'_{i,j} \mid 1 \leq i \leq n+1,\ 0 \leq j \leq n\}\\
&\quad \cup \{A_i, A'_i, B_i, B'_i \mid 1 \leq i \leq n\} \cup \{t, \#\},\\
&R' = \{t_{i,j} \to T_{i+1,j+1},\ T_{i,j} \to T_{i+1,j+1},\ t_{i,j} \to t'_{i,j},\ T_{i,j} \to T'_{i,j},\\
&\qquad t'_{i,j} \to t_{i+1,j+1},\ T'_{i,j} \to T_{i+1,j+1},\ t'_{i,j} \to t_{i+1,j},\ T'_{i,j} \to T_{i+1,j},\\
&\qquad A_i \to A'_i,\ B_i \to B'_i \mid 1 \leq i \leq n\} \cup \{t \to t_{1,0}\ \# \to \#\}\\
&\quad \cup \{T_{i,n+1} \to \# \mid 1 \leq i \leq n\} \cup \{t_{i,n+1} \to \# \mid i \notin M\},\\
&R = \{t_{i,j} \to T_{i+1,j+1}|_{B_i},\ T_{i,j} \to T_{i+1,j+1}|_{B_i},\ t_{i,j} \to t'_{i,j}|_{B'_i},\ T_{i,j} \to T'_{i,j}|_{B'_i},\\
&\qquad t'_{i,j} \to t_{i+1,j+1}|_{A_i},\ T'_{i,j} \to T_{i+1,j+1}|_{A_i},\ t'_{i,j} \to t_{i+1,j}|_{A'_i},\ T'_{i,j} \to T_{i+1,j}|_{A'_i},\\
&\qquad A_i \to A'_i|_{a_i},\ B_i \to B'_i|_{a_i a_i} \mid 1 \leq i \leq n\} \cup \{t \to t_{1,0},\ \# \to \#\}\\
&\quad \cup \{T_{i,n+1} \to \# \mid 1 \leq i \leq n\} \cup \{t_{i,n+1} \to \# \mid i \notin M\}.
\end{aligned}$$

The meaning of $T_{i,n+1}$ is that exactly i input symbols are present, and at least one of them is present multiple times. The meaning of $t_{i,n+1}$ is that the input consisted of exactly i different symbols. This is how an arbitrary finite set is accepted. To accept instead of M its complement, replace $i \notin M$ by $i \in M$ and remove rule $T_{n,n+1} \to \#$. Therefore, deterministic P systems with promoters of weight two accept exactly $NFIN \cup coNFIN$.

For the inhibitor counterpart, notice that the computation of the number of different symbols present, as well as checking if any symbol is present multiple times, stays correct by simply changing promoters to the inhibitors with the same condition, just like in Example 3. Rules processing objects $t_{i,n+1}$ and $T_{i,n+1}$ will have an opposite effect, accepting the complement of the set accepted by the system with promoters, again yielding $NFIN \cup coNFIN$. □

It is still open whether only inhibitors in the rules or only promoters in the rules are sufficient to yield $NFIN \cup coNFIN$ with the asynchronuous mode, too.

4 Conclusion

We have shown the characterizations of deterministic non-cooperative P systems with inhibitors of weight 2, with promoters of weight 2, and with priorities. The first two cases did not reduce the accepting power with respect to unrestricted weight.

References

1. A. Alhazov, R. Freund: Asynchronuous and Maximally Parallel Deterministic Controlled Non-Cooperative P Systems Characterize $NFIN$ and $coNFIN$. *The Tenth Brainstorming Week in Membrane Computing*, vol. 1, Sevilla, 2012, 25–34, and *Membrane Computing - 13th International Conference*, CMC13, Budapest (E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil, Eds.), *Lecture Notes in Computer Science* **7762**, 2013, 101-111.
2. A. Alhazov, D. Sburlan: Ultimately Confluent Rewriting Systems. Parallel Multiset-Rewriting with Permitting or Forbidding Contexts. In: G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa: *Membrane Computing, 5th International Workshop*, WMC 2004, Milano, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, 2005, 178–189.
3. 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.
4. R. Freund, M. Kogler, M. Oswald, A General Framework for Regulated Rewriting Based on the Applicability of Rules. In: J. Kelemen, A. Kelemenová, *Computation, Cooperation, and Life*, Springer, *Lecture Notes in Computer Science* **6610**, 2011, 35–53.

5. R. Freund, S. Verlan: A Formal Framework for Static (Tissue) P Systems. Membrane Computing, 8th International Workshop, WMC 2007 Thessaloniki, 2007, Revised Selected and Invited Papers (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science* **4860**, 2007, 271–284.
6. O.H. Ibarra, H.-C. Yen: Deterministic Catalytic Systems are Not Universal, *Theoretical Computer Science* **363**, 2006, 149–161.
7. M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
8. Gh. Păun: *Membrane Computing. An Introduction*, Springer, 2002.
9. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
10. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
11. P systems webpage. `http://ppage.psystems.eu`

Length P Systems with a Lone Traveler

Artiom Alhazov[1], Rudolf Freund[2], Sergiu Ivanov[3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, MD-2028, Chişinău, Moldova
artiom@math.md
[2] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
rudi@emcc.at
[3] LACL, Université Paris Est, Creteil, Paris, France
sivanov@colimite.fr

Summary. In this paper we consider P systems with linear membrane structures (only one membrane is elementary) with at most one object. We raise and attack the question about the computational power of such systems, depending on the number of membrane labels, kinds of rules used, and some other possible restrictions.

1 Introduction

P systems with symbol objects are formal computational models of parallel distributed multiset processing. In the scope of the present research, we only deal with one object, so the model is reduced to sequential distributed tree rewriting controlled by one traveller object with a finite memory. Moreover, we assume the membrane structure to be linear (the tree is a path), with one or two possible membrane labels. Hence, we are interested in controlled rewriting of strings over one or two symbols.

Unbounded linear membrane structures have received the attention of researchers in the past, see, e.g., [4] and [3]. In the latter paper, the authors spoke about the generation *languages* by representing strings $a_1 \cdots a_n$ as labels membranes arranged in a linear structure as

$$\left[_{a_1} \left[_{a_2} \cdots \left[_{a_n} \right]_{a_n} \cdots \right]_{a_2} \right]_{a_1}.$$

A different example of research where unbounded membrane structures played a crucial role for obtaining an important result (the computational completeness of P systems with active membranes without polarizations) is given in [1], improved in terms of presentation and object/symbol/membrane label complexity in [2].

This research direction, focusing on the membrane structure (rather than the multiset of objects in a designated region) as the result of the computation of a

P system, has been recalled during the 12^{th} Brainstorming Week on Membrane Computing in Sevilla. The technique proposed there how to generate (the description of) recursively enumerable sets of vectors of non-negative integers, using membrane structures with only two labels (0 and 1), where the number of membranes labeled by 1 remains bounded by a constant throughout the computation, is explained in Section 3. It was also conjectured that:

> With one label and at most one "traveler" we can only characterize linear sets, even with membrane generation and deletion.

We confirm this conjecture in Section 4. Finally, in Section 5 we discuss variants of the model leading to weak computational completeness.

2 P Systems with Membrane Creation and Dissolution

A *P system with membrane creation and dissolution* is a construct defined as follows:

$$\Pi = (O, H, \mu, h_1, \cdots, h_n, w_1, \cdots, w_n, R) \text{ where}$$

- O is the (finite) alphabet of *objects*;
- H is the (finite) alphabet of membrane *labels*;
- μ is the initial *membrane structure* consisting of n membranes labeled with elements of H;
- $h_i \in H$, $1 \leq i \leq n$, is the initial label of the membrane i;
- $w_i \in O^*$, $1 \leq i \leq n$, is the string which represents the initial contents of membrane i;
- R is the set of rules.

The rules in R are of one of the following types:

(b) $[_{h_2}\, a [_{h_1}\,]_{h_1}\,]_{h_2} \rightarrow [_{h_2'}\, [_{h_1'}\, b\,]_{h_1'}\,]_{h_2'}$,
$a, b \in O$, $h_1, h_2, h_1', h_2' \in H$ – *send-in* rule,

(c) $[_{h_2}\, [_{h_1}\, a\,]_{h_1}\,]_{h_2} \rightarrow [_{h_2'}\, b [_{h_1'}\,]_{h_1'}\,]_{h_2'}$,
$a, b \in O$, $h_1, h_2, h_1', h_2' \in H$ – *send-out* rule,

(d) $[_{h_2}\, [_{h_1}\, a\,]_{h_1}\,]_{h_2} \rightarrow [_{h_2'}\, b\,]_{h_2'}$,
$a, b \in O$, $h_1, h_2, h_2' \in H$ – *membrane dissolution* rule,

(e) $[_{h_1}\, a\,]_{h_1} \rightarrow [_{h_1'}\, [_{h_2}\, b\,]_{h_2}\,]_{h_1'}$,
$a, b \in O$, $h_1, h_2, h_1' \in H$ – *membrane creation* rule.

A rule of type (b) consumes the symbol a in a membrane with label h_1 and puts a symbol b into an inner membrane, rewriting the labels of the involved membranes. Symmetrically, a rule of type (c) consumes an instance of a in a membrane with label h_1, which is located within a membrane with label h_2, and puts an instance of b into the latter membrane, rewriting the labels.

A rule of type (e) consumes an instance of a in membrane with label h_1 and adds to it a new membrane with label h_1, with an instance of b inside. The label of the original membrane is rewritten to h_1'. Symmetrically, a rule of type (d) consumes an instance of a in a membrane with label h_1, which is located within a membrane with label h_2, copies all the symbols from the membrane with label h_1 to its parent membrane, discards the former membrane, adds a b to the latter membrane and rewrites its label to h_2'.

The rules are applied in the maximally parallel way, with the restriction that in one derivation step at most one rule of types (b), (c), (d), and (e) can be applied per each membrane labeled by h_1 in the definition of the rules given above.

A *configuration* C_k of the system Π consists of the description of the membrane structure μ_k, the labeling of the membranes, and the multisets over O representing the contents of the regions. A configuration is called *halting* if no more rules are applicable any more. A computation of Π is a sequence of configurations $(C_k)_{1\leq k\leq m}$, where C_1 is the initial configuration, C_m is a halting configuration, and C_{k+1} is obtained from C_k by applying the rules from R.

Note: In the following, we will restrict ourselves to P systems with a linear membrane structure. In such systems, membrane creation should never be applied in a non-elementary membrane, otherwise a non-linear membrane structure would be obtained, which is unwanted in the model of the current paper. In Section 3, the P system is constructed in such a way that the object triggering membrane creation would only appear in the elementary membrane. However, in Sections 4 and 5, the setup is more restricted (e.g., one label only), which would yield too restrictive P systems. To overcome this, we impose the following restriction: the membrane creation rule is disabled in the non-elementary membranes. We refer to this rule kind as (e_e). In a similar way, we are interested in membrane dissolution rules which are disabled in the non-elementary membranes, and we denote them by (d_e). Clearly, for each of these membrane dissolution and creation operations, the constructions in Section 3 work for either variant (emphasizing (d_e), otherwise the decrement could be simplified). The regularity conjecture originally assumed only elementary dissolution to be used as well as results only to be obtained at halting with the object in the elementary membrane.

In Section 5 we also consider the following type of rule (introduced already in [3]; we have removed the object in the inner membrane on the right side to let the systems considered in the paper have at most one object during the computation):

(f) $[_{h_1}\, a\,]_{h_1} \to [_{h_1'}\, b[_{h_2}\,]_{h_2}\,]_{h_1'}$,
$a, b \in O$, $h_1, h_2, h_1' \in H$ – *membrane duplication* rule:

the membrane h_1, in the presence of an object a, is duplicated, that is, the label h_1 is changed into h_1', the object a is replaced by b and a new inner membrane labeled by h_2 is created; all the contents of membrane h_1 (membranes or objects except this copy of object b) is now inside membrane h_2.

Note: In [3] the authors have assumed the outer membrane to be the newly created one; it makes no difference as long as we can change both labels by this

rule. However, we prefer to view the *inner* membrane as the new membrane. This lets us keep $h'_1 = h_1$.

Moreover, also the following simplifications/restrictions are made to the rule types in Section 5: membranes h_2, h'_2 are not mentioned in the notation of rules (b), (c) and (d), which means that the rules mentioned above act independently of the external membranes and do not modify them:

(b_r) $a[_{h_1}\,]_{h_1} \to [_{h'_1}\,b]_{h'_1}$, $a, b \in O$, $h_1, h'_1 \in H$,
(c_r) $[_{h_1}\,a]_{h_1} \to b[_{h'_1}\,]_{h'_1}$, $a, b \in O$, $h_1, h'_1 \in H$,
(d_r) $[_{h_1}\,a]_{h_1} \to b$, $a, b \in O$, $h_1 \in H$.

3 Length P Systems

In what follows we will consider a special class of P systems with membrane creation and dissolution. A *length P system* Π is a P system with membrane creation and dissolution which has the following properties:

- the membrane structure is *linear* in every configuration, i.e., every membrane has at most one inner membrane;
- the membranes in any halting configuration of Π are labeled with two labels only.

Consider a halting configuration C of a length P system Π and construct the sequence of membrane labels $(h_i)_{1 \le i \le n}$, in which h_1 corresponds to the label of the skin membrane, h_2 to the label of the membrane inner to the skin membrane, etc., and n is the number of membranes in C. Since h_i is a member of a two-element set, we can interpret this sequence as a vector of numbers coded in unary by runs of one label and separated by instances of the other label. This vector of numbers will be considered as the *output* of the length P system Π.

Following the same convention, we can define the input of Π as a two-label membrane structure coding a certain vector of numbers.

We will now show that length P systems with the input supplied via the membrane structure are computationally complete. To achieve this goal we will pick an arbitrary register machine $\mathcal{M}$ and simulate it with the length P system Π_1 which only uses two labels $H = \{0, 1\}$, only the skin membrane is not empty in the initial configuration and contains q_s, and the sequence of initial membrane labels written as a string $h_1 h_2 \cdots h_n$ has the form $110^{R_1}10^{R_2}1 \cdots 10^{R_m}1$, where R_i, $1 \le i \le m$, is the value of the i-th register of $\mathcal{M}$.

The evolution of the system starts with the rule

$$[_h\, q_s [_1\,]_1\,]_h \to [_h\, [_1\, q_{1,1}\,]_1\,]_h,\ 0 \le h \le 1,$$

where the symbol $q_{1,1}$ represents the first instruction in the program of $\mathcal{M}$ and also keeps the information about the fact that it is located in the region of the membrane structure corresponding to the first register of $\mathcal{M}$.

To simulate an increment of the i-th register of $\mathcal{M}$, we need to add a membrane to the membrane structure and assure that the sequence of labels changes from $110^{R_1}1\cdots10^{R_i}1\cdots10^{R_m}1$ to $110^{R_1}1\cdots10^{R_i+1}1\cdots10^{R_m}1$. We start with the symbol $q_{l,1}$ in the inner membrane of the skin, where l is the label of the increment operation, and we move it to the innermost membrane, counting the registers we traverse on that way:

$$\begin{array}{l}[_h\, q_{l,j}[_0\,]_0\,]_h \to [_h\,[_0\, q_{l,j}\,]_0\,]_h,\\ [_h\, q_{l,j}[_1\,]_1\,]_h \to [_h\,[_1\, q_{l,j+1}\,]_1\,]_h,\ 1 \le j \le m,\ 0 \le h \le 1.\end{array}$$

When we reach the end marker of the last register, we go into the innermost membrane and add a new membrane:

$$[_1\, q_{l,m+1}\,]_1 \to [_1\,[_0\, s_{l,m+1}\,]_0\,]_1.$$

We have now changed the sequence of labels from $110^{R_1}1\cdots10^{R_i}1\cdots10^{R_m}1$ to $110^{R_1}1\cdots10^{R_i}1\cdots10^{R_m}10$.

The series of s-symbols will now swap this new membrane label 0 with the labels of inner membranes, in order to obtain the new 0 label in the zone of the membrane structure corresponding to register R_i:

$$\begin{array}{l}[_0\,[_0\, s_{l,j}\,]_0\,]_0 \to [_0\, s_{l,j}[_0\,]_0\,]_0,\\ [_1\,[_0\, s_{l,j}\,]_0\,]_1 \to [_0\, s_{l,j-1}[_1\,]_1\,]_0,\ i < j \le m+1.\end{array}$$

When we produce the symbol $s_{l,i}$, we have already pushed all labels corresponding to the registers with numbers greater than i towards the innermost membrane and we have added a 0-labeled membrane to the zone corresponding to the i-th register. The following rules move the s-symbol back into the skin membrane:

$$\begin{array}{l}[_h\,[_0\, s_{l,j}\,]_0\,]_h \to [_h\, s_{l,j}[_0\,]_0\,]_h,\\ [_h\,[_1\, s_{l,j}\,]_1\,]_h \to [_h\, s_{l,j-1}[_1\,]_1\,]_h,\ 1 \le j \le i,\ 0 \le h \le 1.\end{array}$$

Finally, $s_{l,0}$ produces the symbol corresponding to the next instruction l' of the register machine:

$$[_h\, s_{l,0}[_1\,]_1\,]_h \to [_h\,[_1\, q_{l',1}\,]_1\,]_h,\ 0 \le h \le 1.$$

The simulation of a decrement and zero-check of the i-th register of $\mathcal{M}$ is symmetric to the simulation of an increment: we start with finding the zone of the membrane structure corresponding to the i-th register:

$$\begin{array}{l}[_h\, q_{l,j}[_0\,]_0\,]_h \to [_h\,[_0\, q_{l,j}\,]_0\,]_h,\\ [_h\, q_{l,j}[_1\,]_1\,]_h \to [_h\,[_1\, q_{l,j+1}\,]_1\,]_h,\ 1 \le j < i,\ 0 \le h \le 1.\end{array}$$

If the value of the register is zero, the symbol $q_{l,i}$ immediately encounters another membrane with the label 1, so it produces the corresponding signal symbol:

$$[_1\, q_{l,i}[_1\,]_1\,]_1 \to [_1\,[_1\, z_{l,i+1}\,]_1\,]_1.$$

The signal symbol then bubbles up into the skin membrane:

$$[_h\,[_0\,z_{l,j}\,]_0\,]_h \to [_h\,z_{l,j}[_0\,]_0\,]_h,$$
$$[_h\,[_1\,z_{l,j}\,]_1\,]_h \to [_h\,z_{l,j-1}[_1\,]_1\,]_h,\ 1 \le j \le i,\ 0 \le h \le 1.$$

Finally, in the outer membranes, $z_{l,0}$ produces the symbol coding the next instruction l', corresponding to unsuccessful decrement:

$$[_h\,z_{l,0}[_1\,]_1\,]_h \to [_h\,[_1\,q_{l',1}\,]_1\,]_h,\ 0 \le h \le 1.$$

If, however, $q_{l,i}$ detects that the i-th register is not empty, it produces a different signal symbol:

$$[_1\,q_{l,i}[_0\,]_0\,]_1 \to [_1\,[_0\,d_{l,i}\,]_0\,]_1.$$

This symbol moves all membrane labels one step outwards:

$$[_0\,d_{l,j}[_0\,]_0\,]_0 \to [_0\,[_0\,d_{l,i}\,]_0\,]_0, \quad i \le j \le m,$$
$$[_0\,d_{l,j}[_1\,]_1\,]_0 \to [_1\,[_0\,d_{l,i+1}\,]_0\,]_1,\ i \le j \le m.$$

When it reaches the end of the zone of the membrane structure corresponding to the m-th register, $d_{l,m+1}$ dissolves the innermost membrane:

$$[_1\,[_0\,d_{l,m+1}\,]_0\,]_1 \to [_1\,s_{l,m+1}\,]_1.$$

Now the s-symbols go outwards into the skin membrane:

$$[_h\,[_0\,s_{l,j}\,]_0\,]_h \to [_h\,s_{l,j}[_0\,]_0\,]_h,$$
$$[_h\,[_1\,s_{l,j}\,]_1\,]_h \to [_h\,s_{l,j-1}[_1\,]_1\,]_h,\ 1 \le j \le m+1,\ 0 \le h \le 1.$$

And finally, $s_{l,0}$ generates the symbol coding the next operation l'', corresponding to a successful decrement:

$$[_h\,s_{l,0}[_1\,]_1\,]_h \to [_h\,[_1\,q_{l'',1}\,]_1\,]_h,\ 0 \le h \le 1.$$

Note: The construction can be rewritten such that only one membrane participates on the left side of any rule, but then the total number of membranes labeled 1 will no longer remain a constant, but will still stay bounded. In the construction presented above, however, the number of membranes labeled by 1 is constant during the computation, namely, it is $m+2$. In the construction above, membrane creation and membrane dissolution rules do not modify the label of the outer membrane, while the communication rules either keep unchanged the labels of the two membranes they work with, or they swap them. The number of membranes labeled by 1 may be reduced by one, by starting with the skin labeled by 0; this does not affect the proof. Moreover, with a technique mentioned in the end of Section 5 the innermost membrane labeled 1 may be avoided. For the special case $m = 2$, in Section 5 we present a construction when, using membrane duplication, we avoid even the membrane labeled by 1 which separates the representation of the two registers, by keeping track of this position with the object itself.

Before that, in the following section, we proceed with the conjecture that length P systems with one label only generate regular sets of numbers, in case of elementary membrane creation, elementary membrane dissolution and communication rules.

4 On the Regularity Conjecture

Clearly, any regular set of numbers can be generated with rules (e) only, simulating each rule $pa \to q$, $p \to q_h$ of a finite automaton (for which, without restricting generality we require that every state except q_h is non-final and has at least one outgoing transition) by rules $[_0\, p\,]_0 \to [_0\, [_0\, q\,]_0\,]_0$ and $[_0\, p\,]_0 \to [_0\, [_0\, q_h\,]_0\,]_0$, respectively. Here, the skin and the elementary membrane are to be seen as additional endmarkers (as otherwise only numbers ≥ 2 could be generated). The latter can be avoided by additionally using a rule of type (d) $[_0\, [_0\, q_h\,]_0\,]_0 \to [_0\, q'_h\,]_0$.

There are two possible reasons why the power of length P systems with one label and one object is restricted. The first reason (R1) is that the object can never detect that it is in the skin (unless we additionally allow the skin to have a distinguished label). Indeed, if $C \Rightarrow C'$ is one computation step, then it is easy to see (just by looking at all kinds of rules) that

$$[_0\, C\,]_0 \Rightarrow [_0\, C'\,]_0 \tag{1}$$

is also a valid computation step. This immediately generalizes to multiple membrane levels and multiple derivation steps:

$$C \Rightarrow^* C' \longrightarrow \underbrace{[_0 \cdots [_0}_{n} C\,]_0 \cdots\,]_0 \Rightarrow^* \underbrace{[_0 \cdots [_0}_{n} C'\,]_0 \cdots\,]_0. \tag{2}$$

The second reason (R2) is that the total membrane depth can only be increased from the elementary membrane (unless we additionally allow membrane duplication rules). Let us denote by $C \Rightarrow_{\text{dive}} C'$ a "membrane dive", i.e., such a computation fragment that the object is in the elementary membrane in both C and in C', but never in the intermediate configurations. Let us also denote by $\langle n, a\rangle$ the configuration consisting of a linear structure of n membranes, the elementary one containing object a:

$$\langle n, a\rangle = \underbrace{[_0 \cdots [_0}_{n} a\,]_0 \cdots\,]_0.$$

Clearly, for any $a, b \in O$, one of the following cases is true:

- There exists some minimal value $n \in \mathbb{N}$ for which $\langle n, a\rangle \Rightarrow_{\text{dive}} \langle n, b\rangle$ is true. Let us denote this value by $n(a, b)$. We recall from (2) that $\langle n, a\rangle \Rightarrow_{\text{dive}} \langle n, b\rangle$ holds for any $n \geq n(a, b)$.
- $\langle n, a\rangle \Rightarrow_{\text{dive}} \langle n, b\rangle$ is not true for any $n \in \mathbb{N}$.

We denote by $\bar{n}$ the maximum of values $n(a, b)$ over the first case. We denote by N the set of all numbers generated by the length P system not exceeding $\bar{n}$. We denote by A the set of all objects $a \in O$ such that $\langle \bar{n}, a\rangle$ is reachable in the length P system.

It is easier to confirm the conjecture for the case when the membrane dissolution is only allowed for the elementary membranes. Then, the power of the length

P system is described by the union of the finite set N with the power (plus number N) of a partially blind 1-register machine starting in states from A, where the chain rules $a \to b$ correspond to the relation $\langle \bar{n}, a \rangle \to \langle \bar{n}, b \rangle$, and the increment/decrement instructions are associated to the rules creating and dissolving elementary membranes. It is known, see, e.g., [5], that the number sets generated by partially blind 1-register machines is $NMAT = NREG$. Finally, in this case it is immediate that the membrane structure cannot change between the last time the object is in the elementary membrane and the halting.

Note: The regularity conjecture remains valid even if non-elementary membranes can be dissolved. Indeed, a similar (non-constructive) argument can be made; we only describe it *informally*. For any two symbols $a, b \in O$, either there exists some minimal number $m \in \mathbb{N}$ such that $\langle m, a \rangle \Rightarrow \langle m+1, b \rangle$, and we denote it by $m(a, b)$, or this derivation is not possible for any $m \in \mathbb{N}$. Then the behaviour of the length P system can be decomposed into a finite part (the membrane structure depth not exceeding the maximum of all defined values $m(a, b)$), and a regular part defined by the binary relation over O which is the domain where $m(a, b)$ is defined. As for the computation between the last time the object is in the elementary membrane and the halting (in case the halting is not in the elementary membrane), this should also preserve regularity, because without being able to test for skin and without returning to the elementary membrane, we just have a finite control walking across the unilabeled membrane structure, and membrane dissolution in this case can only perform some regular erasing transformation.

5 Weak Computational Completeness

In this section we show that we can construct length P systems with one label which are weakly computationally complete, assuming the following ingredients:

- Membrane duplication rules are allowed (nullifying reason R2).
- The skin can be distinguished (nullifying reason R1) either by being the only membrane with another label ($s = 1$), or by having its own set of rules R_s. We use the first case for the presentation of the result.
- As typical in membrane computing, the dissolution is not limited to elementary membranes.
- Membrane creation rules are disabled in the non-elementary membranes. (Alternatively, if one allows to also have special rules for the elementary membrane, we may forbid membrane creation in the rest of the system).

The result relies on simulating 2-register machines, storing the register values in the multiplicity of membranes, using the object to separate the two numbers.

$$[_1 \underbrace{[_0 \cdots [_0}_{n_1} a \underbrace{[_0 \cdots [_0}_{n_2}]_0 \cdots]_0]_0 \cdots]_0]_1 \tag{3}$$

We first present a simpler construction, assuming an additional elementary membrane labeled 1. In this case, membrane creation rules are not even needed.

Indeed, the first register is tested for zero by using the skin rules (duplicate, enter, dissolve). The second register is tested for zero by entering one membrane, checking its label and exiting it.

The increment is done by a duplication rule (f), in case of the first register followed by entering the newly created membrane.

The decrement is performed by a dissolution rule (d), in case of the second register preceded by moving the object into the next membrane.

We proceed to the formal description of the simulation (membrane label h stands for any of 0 or 1).

$(l_1 : A(1), l_2, l_3)$ is performed as:

$[_h\, l_1\,]_h \to [_h\, l_1'[_0\,]_0\,]_h,\ l_1'[_0\,]_0 \to [_0\, l_2\,]_0, \qquad l_1'[_0\,]_0 \to [_0\, l_3\,]_0.$

$(l_1 : A(2), l_2, l_3)$ is performed as:

$[_h\, l_1\,]_h \to [_h\, l_2[_0\,]_0\,]_h,\ [_h\, l_1\,]_h \to [_h\, l_3[_0\,]_0\,]_h.$

$(l_1 : S(1), l_2, l_3)$ is performed as:

$[_0\, l_1\,]_0 \to l_2,$

$[_1\, l_1\,]_1 \to [_1\, l_1'[_0\,]_0\,]_1,\ l_1'[_0\,]_0 \to [_0\, l_1''\,]_0, \qquad [_0\, l_1''\,]_0 \to l_3.$

$(l_1 : S(2), l_2, l_3)$ is performed as:

$l_1[_0\,]_0 \to [_0\, l_1'\,]_0, \qquad [_0\, l_1'\,]_0 \to l_2,$

$l_1[_1\,]_1 \to [_1\, l_1'\,]_1, \qquad [_1\, l_1'\,]_1 \to l_3[_1\,]_1.$

Notice that the first three operations above ($A(1)$, $A(2)$ and $S(1)$) operate identically also in the absence of the elementary membrane labeled 1, and so does the first line of $S(2)$, corresponding to the decrement case. Using membrane creation (e) (immediately followed by dissolution) to test for the membrane elementarity, it is possible to avoid the extra elementary membrane with 1, and stay with the representation (3): we replace the last two rules of the construction above with the following ones:

$$[_h\, l_1\,]_h \to [_h\, [_0\, l_1''\,]_0\,]_h,\ [_0\, l_1''\,]_0 \to l_3.$$

In this way, using only one label for non-skin membranes, weak computational completeness of length P systems is shown with one object, by taking advantage of rules creating non-elementary membranes under the assumption that elementary membrane creation is disabled in the non-elementary membranes.

6 Discussion

We have introduced P systems with a linear membrane structure (i.e., only one membrane is elementary) with at most one object. The result of such systems,

called length P systems, is either the total number of membranes at halting, or the vector of numbers of consecutive membranes labeled 0. In Section 3 we presented the simulation of register machines with any fixed number of registers.

The power of length P systems with one object and one membrane label depends on two factors: whether the object can detect being in the skin membrane, and whether non-elementary membrane creation is allowed. The first factor is related to the zero-test of the "first" register, and the second factor is related to the possibility of effectively operating with two numbers instead of one. Since the regularity conjecture assumed that these two ingredients are not allowed, we have *confirmed* the conjecture.

In Section 5 we have shown that removing both of these conditions leads to P systems being weakly computationally complete. Questions arise about the intermediate extensions.

We now formulate the following the following two *conjectures*:

Conjecture 1. Length P systems produce only regular languages even when membrane duplication is allowed (i.e., without reason R2). The intuition behind the conjecture is that such P systems relate to 2-register machines, where one register is blind (i.e., it can be incremented and decremented, but cannot be tested for zero, and the machine's computation is discarded without the result if decrement of zero is attempted).

Conjecture 2. Length P systems produce only regular languages even if the skin membrane has a distinguished label (i.e., without reason R1). The intuition behind the conjecture is that such P systems should relate to 1-register machines, but for the proof many more cases would have to be investigated.

In an *extended version* of this paper we plan to consider length P systems where the number of membranes labeled by 1 is not bounded, considering binary strings as the results instead of vectors.

Acknowledgements. Artiom Alhazov acknowledges project STCU-5384 *Models of high performance computations based on biological and quantum approaches* awarded by the Science and Technology Center in the Ukraine.

References

1. A. Alhazov: P Systems without Multiplicities of Symbol-Objects. *Information Processing Letters* **100** (3), 2006, 124–129.
2. A. Alhazov, R. Freund, A. Riscos-Nunez: Membrane Division, Restricted Membrane Creation and Object Complexity in P Systems. *International Journal of Computer Mathematics* **83** (7), 2006, 529-548.
3. F. Bernardini, M. Gheorghe: Languages Generated by P Systems with Active Membranes. *New Generation Computing* **22** (4), 2004, 311–329.
4. R. Freund: Special Variants of P Systems Inducing an Infinite Hierarchy with Respect to the Number of Membranes. *Bulletin of the EATCS* **75**, 2001, 209–219.

5. R. Freund, O.H. Ibarra, Gh. Păun, H.C. Yen: Matrix Languages, Register Machines, Vector Addition Systems. Proceedings of the *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–167.

Life-Death Ratio Approach by a Multiset-Based Type System

Bogdan Aman, Gabriel Ciobanu

Romanian Academy, Institute of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Summary. We introduce and study a multiset-based type system with ratio thresholds motivated by an important regulatory mechanism inside a cell which try to maintain a "life-death" ratio between some given lower and upper thresholds. We use such a type system to control ratio thresholds in a bio-inspired and multisets-based formalism. For this type system we prove a subject reduction theorem, together with soundness and completeness theorems. A type inference for deducing the type of a system is presented.

1 Introduction

Membrane systems have been introduced as a computational model inspired by cellular biology [9], and have been later applied to the description of biological systems [6]. Possible links between process calculi and membrane systems are presented in [7].

Membrane systems usually consider cells as mechanisms working in a maximal parallel and non-deterministic manner.

However, the living cells do not work in such ways; a chemical reaction takes place only if certain constraints are fulfilled (e.g, certain ratios are between given thresholds in sodium/potassium pump [3] and ratio-dependent predatorprey systems [8]). In order to cope with such constraints, in [2] we enriched the membrane systems with integral proteins by adding a quantitative type discipline. We associated to each system a set of constraints that must be satisfied in order to assure that the application of the rules to a well-formed membrane system leads to a well-formed membrane system as well. We think that this two-stage approach to the description of biological behaviours is of interest, where the first describes reactions in an "untyped" setting, and then rules out certain evolutions by imposing thresholds. This allows one to treat separately different aspects of the modelling: what transitions are possible at all, and under which circumstances they can take place. In this paper we provide a type inference algorithm for deducing the type of a system.

The type system for membrane systems with integral proteins follows the research line started in [1] where a type system for membrane system with symport/antiport rules is presented. The work is also related to [4] where a type systems for the calculus of looping sequences is defined based on the number of elements and not on the ratios between elements. The presentation of the typed sodium/potassium pump in [2] is used as a motivation and a running example for typing the membrane systems. This paper is related to our previous article [2].

2 Membrane Systems with Integral Proteins

Membrane systems are parallel and nondeterministic computing models inspired by the compartments of cells and their biochemical reactions. The structure of the cell is represented by a set of hierarchically embedded regions, each one delimited by a surrounding boundary *called membrane*, and all of them contained inside an external special region called the *skin* membrane. Multisets of objects are distributed inside these regions, and they can be modified or moved between adjacent compartments. Objects represent the formal counterpart of the molecular species (ions, proteins, etc.) floating inside cellular compartments, and they are described by means of strings over a given alphabet. Evolution rules represent the formal counterpart of chemical reactions, and are given in the form of rewriting rules that operate on the objects, as well as on the compartmentalised structure (e.g., by dissolving, dividing, creating, or moving membranes).

Starting from an initial configuration, the multisets of objects initially placed inside the compartmentalised structure, the system evolves by applying the evolution rules in a nondeterministic and maximally parallel manner. A rule is applicable when all the objects appearing on its left hand side are available in the region where the rule is placed. The maximal parallelism of rule application means that each applicable rule that is inside a region *has to* be applied in that region. Since there is a competition for the available objects, only certain (nondeterministically selected) rules are applied. A halting configuration is reached when no rule is applicable, and the result is given by the number of objects (in a specified region). More details can be found in [9, 10].

In what follows we present some technical notions used in this paper. Given a finite set O of symbols, the set of all strings over O is denoted by O^*, and the set of all non-empty strings over O, is denoted by $O^+ = O^* \backslash \lambda$, where λ is the empty string. A multiset over O is a map $u : O \to \mathbb{N}$, where $u(a)$ denotes the multiplicity of the symbol $a \in O$ in the multiset u and $|u| = \sum_{a \in O} u(a)$ denotes the number of objects appearing in the multiset u. We say that a multiset u is included into a multiset v (denoted by $u \subseteq v$) if $u(a) \leq v(a)$ for all $a \in O$. The empty multiset is denoted by ϵ, and satisfies $\epsilon(a) = 0$, for all $a \in O$. For two multisets u and v we define the sum $u + v$ by $(u + v)(a) = u(a) + v(a)$ for all $a \in O$, and the difference $u - v$ by $(u - v)(a) = max\{0, u(a) - v(a)\}$ for all $a \in O$. More details can be found in [11].

Inspired by [5], we present a membrane system that is able to model biological pumps, in which exist attachment/de-attachment of objects to/from the integral proteins of the membranes, and transformation of objects inside a region if certain integral proteins are present in the surrounding membranes. To each membrane is associated a label $i \in Lab$, and two multisets s_i and v_i over O^*, and the membrane is denoted by $[\ s_i\ ]^i_{v_i}$. If s_i and/or v_i are the empty multisets, they are omitted.

Definition 1. *A* membrane system with integral proteins of degree n *is:*

$$\prod = (O, Lab, \mu, v_1/s_1, \ldots, v_n/s_n, R),\ \textit{where:}$$

1. O *is an alphabet (finite, non-empty) of objects;*
2. L *is a finite set of labels;*
3. μ *is a membrane hierarchical structure with* $n \geq 2$ *membranes;*
4. v_i, $1 \leq i \leq n$ *is a multiset of integral proteins present on the membrane* i*;*
5. s_i, $1 \leq i \leq n$ *is a multiset of objects placed inside the membrane* i*;*
6. R *is a finite set of rules of the following forms:*

a) $[\ \alpha\ ]^i_v \rightarrow [\ ]^i_{v'}$, $\alpha \in O^+$, $v, v' \in O^*$, $i \in Lab$; $(attach_{in})$

This rule is applicable if the following conditions are fulfilled:

- *the multiset* α *exists inside region* i $(\alpha \subseteq s_i)$*;*
- *the multiset* v *of integral proteins exists on membrane* i $(v \subseteq v_i)$*.*

When this rule is applied to membrane i*:*

- α *is removed from* s_i*;*
- v *is modified to* v'*;*
- *the objects not involved in this rule are left unchanged.*

b) $[\ [\ ]^i_v\ \alpha]^j \rightarrow [\ [\ ]^i_{v'}]^j$, $\alpha \in O^+$, $v, v' \in O^*$, $i, j \in Lab$; $(attach_{out})$

This rule is applicable if the following conditions are fulfilled:

- *the multiset* α *exists inside region* j $(\alpha \subseteq s_j)$*;*
- *the multiset* v *of integral proteins exists on membrane* i $(v \subseteq v_i)$*.*

When this rule is applied to membrane i *we have:*

- α *is removed from* s_j*;*
- v *is modified to* v'*;*
- *the objects not involved in this rule are left unchanged.*

c) $[\ ]^i_v \rightarrow [\ \alpha\ ]^i_{v'}$, $\alpha \in O^+$, $v \in O^+$, $v' \in O^*$, $i \in Lab$; $(de-attach_{in})$

This rule is applicable if the following conditions are fulfilled:

- *the multiset* v *of integral proteins exists on membrane* i $(v \subseteq v_i)$*.*

When this rule is applied to membrane i*:*

- v *is modified to* v'*;*
- α *is added to* s_i*;*
- *the objects not involved in this rule are left unchanged.*

d) $[\ [\ ]^i_v]^j \rightarrow [\ [\ ]^i_{v'}\ \alpha]^j$, $\alpha \in O^+$, $v \in O^+$, $v' \in O^*$, $i, j \in Lab$; $(de-attach_{out})$

This rule is applicable if the following conditions are fulfilled:

- *the multiset* v *of integral proteins exists on membrane* i $(v \subseteq v_i)$*.*

When this rule is applied to membrane i *we have:*

- v *is modified to* v'*;*

- *α is added to s_j;*
- *the objects not involved in this rule are left unchanged.*

e) $[\ \alpha\ ]^i_v \to [\beta\ ]^i_v$, $v, \beta \in O^*$, $\alpha \in O^+$, *and* $i \in Lab$. (*local evol*)

This rule is applicable if the following conditions are fulfilled:

- *the multiset α exists inside region i ($\alpha \subseteq s_i$);*
- *the multiset v of integral proteins exists on membrane i ($v \subseteq v_i$).*

When this rule is applied to membrane i:

- *α is removed from s_i;*
- *β is added to s_i;*
- *the objects not involved in this rule are left unchanged.*

The way in which the evolution rules are applied is detailed in what follows. In order to formally represent the configurations of membrane systems with integral proteins, we define terms ranged over by st, st_1, ..., that are built by means of a membrane constructor $[-]^-_-$, using a set O of objects and a set Lab of labels. The syntax of the terms $st \in ST$ is given by

$$st ::= u \mid [st]^i_v \mid st\ st\ ,$$

where u denotes a (possibly empty) multiset of objects placed inside a membrane, v a multiset of objects within or on the surface of a membrane, i a membrane label, and $st\ st$ is the parallel composition of two terms. Since we work with multisets of terms, we introduce a structural congruence relation following a standard approach from process algebra. The defined structural congruence is the least congruence relation on terms satisfying also the rule:

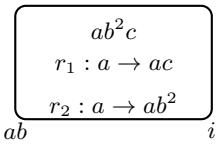
$$\text{if } v_1 \equiv v_2 \text{ and } st_1 \equiv st_2 \text{ then } [st_1]^i_{v_1} \equiv [st_2]^i_{v_2}.$$

A pattern is a term that may include variables. We denote by $\mathcal{P}$ the infinite set of patterns P of the form: $P ::= st \mid [\ P\ X\ ]^i_{v\ y} \mid P\ P$. We distinguish between "simple variables" (ranged over by x, y, z) that may occur only on the surface of membranes (i.e., they can be replaced only by multisets of objects) and "term variables" (ranged over by X, Y, Z) that may only occur inside regions (they can be replaced by arbitrary terms). Therefore, we assume two disjoint sets: V_{O^*} (set of simple variables) and V_{ST^*} (set of term variables). We denote by $V = V_{O^*} \cup V_{ST^*}$ the set of all variables, and with ρ any variable in V.

An instantiation is a partial function $\sigma : V \to ST^*$ that preserves the type of all variables: simple variables ($x \in V_{O^*}$) and term variables ($X \in V_{ST^*}$) are mapped into objects ($\sigma(x) \in O^*$) and terms ($\sigma(X) \in ST^*$), respectively. Given a pattern P, the term obtained by replacing all occurrences of each variable $\rho \in V$ with the term $\sigma(\rho)$ is denoted by $P\sigma$. The set of all possible instantiations is denoted by Σ, and the set of all variables appearing in P is denoted by $Var(P)$.

Formally, a rewriting rule r is a pair of patterns (P_1, P_2), denoted by $P_1 \to P_2$, where $P_1 \neq \epsilon$ (i.e., P_1 is a non-empty pattern) and $Var(P_2) \subseteq Var(P_1)$. A rewriting rule $P_1 \to P_2$ states that a term $P_1\sigma$ can be transformed into the term $P_2\sigma$, for some instantiation function σ.

Example 1. Consider the membrane system depicted in what follows. In the right part of the picture we give some examples of the notions defined above.



Terms: $[ab^2c]^i_{ab}$ or ab^2c
Patterns: $[aX]^i_y$ or aX
Instantiation: $\sigma(X) = b^2c$, $\sigma(y) = ab$
Rewriting rule r_1: $aX \to acX$
Rewriting rule r_2: $aX \to ab^2X$

It can be noticed that using the given instantiations the pattern $[aX]^i_y$ becomes the term $[ab^2c]^i_{ab}$, while the pattern aX becomes the the term ab^2c. The rewriting rules are a generalization of the rules used in P systems, by adding variables that stand for the objects not involved in the evolution of a system.

The notion of context is used to complete the definition of a rewriting semantics for our systems. This is done by enriching the syntax with a new object $\Box$ representing a hole. By definition, a context is represented as a single hole $\Box$. The infinite set $\mathcal{C}$ of contexts (ranged over by C) is given by:

$$C ::= \Box \mid C\ st \mid [\ C\ ]^i_v.$$

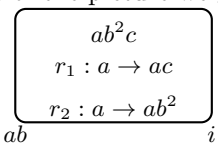
Given $C_1, C_2 \in \mathcal{C}$, $C_1[\ st\ ]$ denotes the term obtained by replacing $\Box$ with st in C_1, while $C_1[\ C_2\ ]$ denotes the context obtained by replacing $\Box$ with C_2 in C_1.

Given a set R of rewriting rules, a reduction semantics of the system is given by the least transition relation $\to$ closed with respect to $\equiv$ satisfying also the rule:

$$\frac{P_1 \to P_2 \in R \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma \quad C \in \mathcal{C}}{C[P_1\sigma] \to C[P_2\sigma]}\ .$$

$\to^*$ denotes the reflexive and transitive closure of $\to$.

Example 2. Consider the membrane system depicted in what follows. In the right part of the picture we give some examples of the notions defined previously.



Rewriting rule r_1: $aX \to acX$
Instantiation: $\sigma(X) = b^2c$
Context: $[\Box]^i_{ab}$
Transition: $[ab^2c]_{ab} \to [ab^2c^2]_{ab}$

It can be noticed that using the given instantiation and context the rewriting rule r_1: $aX \to acX$ can be used to perform the transition $[ab^2c]_{ab} \to [ab^2c^2]_{ab}$, thus modelling the application of rule r_1 is the above membrane system.

3 Threshold-Based Type System Over Multisets

We use the type system defined in [2]. Let T be a finite set of basic types ranged over by t. We classify each object in O with a unique element of T; we use Γ to denote this classification. In general, different objects a and b can have the same basic type t. When there is no ambiguity, we denote the type associated with an object a by t_a. For each ordered pair of basic types (t_1, t_2), we assume the existence of two functions, $min : T \times T \to (0, \infty) \cup \{\diamond\}$ and $max : T \times T \to (0, \infty) \cup \{\diamond\}$. These functions indicate the minimum and maximum ratio between the number of objects of basic types t_1 and t_2 that can be present inside a membrane.

Definition 2 (Consistent Basic Types). *A system using a set of basic types T and the functions min and max is consistent if:*

1. $\forall t_1, t_2 \in T$, $min(t_1, t_2) \neq \diamond$ *iff* $max(t_1, t_2) \neq \diamond$;
2. $\forall t_1, t_2 \in T$, $min(t_1, t_2) \neq \diamond$ *iff* $min(t_2, t_1) \neq \diamond$;
3. $\forall t_1, t_2 \in T$ *if* $min(t_1, t_2) \neq \diamond$, *then* $min(t_1, t_2) \leq max(t_1, t_2)$;
4. $\forall t_1, t_2 \in T$ *if* $min(t_1, t_2) \neq \diamond$ *and* $max(t_2, t_1) \neq \diamond$, *then* $min(t_1, t_2) \cdot max(t_2, t_1) = 1$.

Example 3. Consider $T = \{t_a, t_b, t_c\}$ and min, max defined as:

$min(t_1, t_2)$

$t_1 \backslash t_2$	t_a	t_b	t_c
t_a	$\diamond$	0.4	$\diamond$
t_b	0.2	$\diamond$	1/6
t_c	$\diamond$	3	$\diamond$

$max(t_1, t_2)$

$t_1 \backslash t_2$	t_a	t_b	t_c
t_a	$\diamond$	5	$\diamond$
t_b	2.5	$\diamond$	1/3
t_c	$\diamond$	6	$\diamond$

The system is consistent since each pair of types respects Definition 2.

Definition 3. Quantitative types *are triples (L, Pr, U) over the set T of basic types, where L (lower) is the set of minimum ratios between basic types, Pr (present) is the multiset of basic types of present objects (the objects present at the top level of a pattern, i.e. in the outermost membrane), and U (upper) is the set of maximum ratios between basic types.*

The number of objects of type t appearing in a multiset Pr is denoted by $Pr(t)$. In order to define well-formed types, given a multiset M of types, the sets RP_M (ratios of present types in M), L_M (lower bounds of present types in M) and U_M (upper bounds of present types in M) are required:

- $RP_M = \begin{cases} \emptyset & \text{if } |M| \leq 1 \\ \bigcup_{t,t' \in M} \left\{ t/t' : \dfrac{Pr(t)}{Pr(t')} \mid t \neq t', Pr(t') \neq 0 \right\} & \text{otherwise} \end{cases}$
- $L_M = \begin{cases} \emptyset & \text{if } |M| \leq 1 \\ \bigcup_{t,t' \in M} \{ t/t' : min(t, t') \mid t \neq t', min(t, t') \neq \diamond \} & \text{otherwise} \end{cases}$
- $U_M = \begin{cases} \emptyset & \text{if } |M| \leq 1 \\ \bigcup_{t,t' \in M} \{ t/t' : max(t, t') \mid t \neq t', max(t, t') \neq \diamond \} & \text{otherwise} \end{cases}$

These sets contain labelled values in order to be able to refer to them when needed: e.g., $t/t' : \dfrac{Pr(t)}{Pr(t')}$ denotes the fact that the ratio between the objects of types t and t' that are present in Pr has the label t/t', and the value is $\dfrac{Pr(t)}{Pr(t')}$.

Definition 4 (Well-Formed Types). *A type (L, Pr, U) is well-formed if*
$$L = L_{Pr},\ U = U_{Pr} \text{ and } L \leq RP_{Pr} \leq U.$$

Remark 1. If the set T contains a large number of basic types, defining a type to be well-formed only if $L = L_{Pr}$ and $U = U_{Pr}$ reduces the amount of information encapsulated by a type. E.g., for $|T| = 100$, the number of entries in the *min* table is equal to 10000.

Example 4. Let us assume a set of basic types $T = \{t_a, t_b\}$, a classification $\Gamma = \{a : t_a, b : t_b\}$ and the functions *min*, *max* defined as:

$min(t_1, t_2)$

$t_1 \backslash t_2$	t_1	t_2
t_1	$\diamond$	0.5
t_2	0.3	$\diamond$

$max(t_1, t_2)$

$t_1 \backslash t_2$	t_1	t_2
t_1	$\diamond$	10/3
t_2	2	$\diamond$

The term a^5b^2 is well-formed, while the term a^9b is not, because the ratio between t_a and t_b equals 9, and so it exceeds the maximum 10/3 indicated in *max* table.

From now on we work only with well-formed types. For instance, the two well-formed types (L, Pr, U) and (L', Pr', U') of the following two definitions are constructed by using specific ratio tables for *min* and *max*.

Definition 5 (Type Compatibility). *Two well-formed types* (L_{Pr}, Pr, U_{Pr}) *and* $(L_{Pr'}, Pr', U_{Pr'})$ *are compatible, written* $(L, Pr, U) \bowtie (L_{Pr'}, Pr', U_{Pr'})$, *if* $L_{Pr+Pr'} \leq RP_{Pr+Pr'} \leq U_{Pr+Pr'}$.

Example 5 (cont.). Consider the assumptions from Example 4. The types $(L_{t_a^5\ t_b^2}, t_a^5 t_b^2, U_{t_a^5\ t_b^2})$, $(\emptyset, t_a, \emptyset)$ and $(\emptyset, t_a^4, \emptyset)$ are well-formed. It holds that

$$(L_{t_a^5 t_b^2}, t_a^5 t_b^2, U_{t_a^5\ t_b^2}) \bowtie (\emptyset, t_a, \emptyset) \text{ and } (L_{t_a^5 t_b^2}, t_a^5\ t_b^2, U_{t_a^5\ t_b^2}) \not\bowtie (\emptyset, t_a^4, \emptyset),$$

since the ratio in the second case between t_a and t_b equals 4.5, and so it exceeds the maximum 10/3 indicated in *max* table.

A **basis** Δ assigning types to simple and term variables is defined by

$$\Delta ::= \emptyset \mid \Delta, x : (L_t, t, U_t) \mid \Delta, X : (L, Pr, U).$$

A basis is well-formed if all types in the basis are well-formed.

A **classification** Γ maps each object in O to a unique element of the set T of basic types. The judgements are of the form $\Delta \vdash P : (L, Pr, U)$ indicating that a pattern P is well-typed having the type (L, Pr, U) relative to a typing environment Δ.

Types are assigned to patterns and terms according to the typing rules of Table 1. It is not difficult to verify that a derivation starting from well-formed bases produces only well-formed bases and well-formed types.

We define a typed semantics, since we are interested in applying reduction rules only to correct terms having well-formed types, and whose requirements are satisfied. More formally, a term *st* is correct if $\emptyset \vdash st : (L, Pr, U)$ for some well-formed type (L, Pr, U). An instantiation σ agrees with a basis Δ (denoted by $\sigma \in \Sigma_\Delta$) if $\rho : (L, Pr, U) \in \Delta$ implies $\emptyset \vdash \sigma(\rho) : (L, Pr, U)$.

In order to apply the rules in a safe way, we introduce a restriction on rules based on the context of application rather than on the type of patterns involved in

Table 1. Typing Rules

$$\Delta \vdash \epsilon : (\emptyset, \emptyset, \emptyset) \quad (TEps) \qquad \frac{a : t \in \Gamma}{\Delta \vdash a : (L_t, t, U_t)} \quad (TObj)$$

$$\Delta, \rho : (L, Pr, U) \vdash \rho : (L, Pr, U) \quad (TVar)$$

$$\frac{\Delta \vdash v : (L, Pr, U) \quad \Delta \vdash P' : (L', Pr', U') \quad (L, Pr, U) \bowtie (L', Pr', U') \quad i \in Lab}{\Delta \vdash [P']_v^i : (L, Pr, U)} \quad (TMem)$$

$$\frac{\Delta \vdash P : (L_{Pr}, Pr, U_{Pr}) \quad \Delta \vdash P' : (L_{Pr'}, Pr', U_{Pr'}) \quad (L_{Pr}, Pr, U_{Pr}) \bowtie (L_{Pr'}, Pr', U_{Pr'})}{\Delta \vdash P\ P' : (L_{Pr+Pr'}, Pr + Pr', U_{Pr+Pr'})} \quad (TPar)$$

the rule. In this direction, we characterise contexts by the types of terms that can fill their hole, and the rules by the types of terms produced by their application.

Definition 6 (Typed Holes). *Given a context C and a type (L, Pr, U) that is well-formed, the type (L, Pr, U)* **fits the context** *C if for some well-formed type (L', Pr', U') it can be shown that $X : (L, Pr, U) \vdash C[X] : (L', Pr', U')$.*

Example 6 (cont.). Consider the notions from Example 4, and also

- a term $[a^5\ b^4]a^{12}\ b^3$,
- a rule $r_1 : [\ a^3\ X\ ]_x \to [\ X\ ]_{a^3\ x}$ (having the form $P_1 \to P_2$),
- a context $C = \Box\ a^{12}\ b^3$,
- instantiations $\sigma(X) = a^2\ b^4$, $\sigma(x) = \epsilon$.

By applying rule r_1 the term $[a^2\ b^4]_{a^3}a^{12}\ b^3$ is obtained. This is not well-typed since the type $(L_{t_a^3}, t_a^3, U_{t_a^3})$ of P_2 does not fit the context C. It does not fit since the term $C[P_2]$ has the type $(L_{t_a^{15}t_b^3}, t_a^{15}t_b^3, U_{t_a^{15}t_b^3})$ that is not well-formed due to the fact that the ratio between t_a and t_b at the top level is $\frac{12+3}{3} = 5$ that is greater than $10/3$.

The above notion guarantees that we obtain a correct term filling a context with a term whose type fits the context: note that there may be more than one type (L, Pr, U) such that (L, Pr, U) fits the context C.

We can classify reduction rules according to the types that can be derived for the right hand sides of the rules, since they influence the type of the obtained term.

Definition 7 (Δ-(L, Pr, U) safe rules). *A rewriting rule $P_1 \to P_2$ is Δ safe if for some well-formed type (L, Pr, U) it can be shown that $\Delta \vdash P_2 : (L, Pr, U)$.*

To ensure correctness, each application of a rewriting rule must verify that the type of the right hand side of the rule fits the context. Using Definitions 6 and 7, if it is applied a rule whose right hand side has type (L, Pr, U) and this type fits the context, then a correct term is obtained.

Typed Semantics.

Given a finite set R of rewriting rules (e.g., the one presented in Table 2), the typed semantics of a system is given by the least relation $\Rightarrow$ closed with respect to $\equiv$ and satisfying the following rule:

$$\frac{\begin{array}{c} P_1 \to P_2 \in R \text{ is a } \Delta\text{-}(L, Pr, U) \text{ safe rule, } P_1\sigma \not\equiv \epsilon \\ \sigma \in \Sigma_\Delta \qquad C \in \mathcal{C} \qquad \text{and} \qquad (L, Pr, U) \text{ fits } C \end{array}}{C[P_1\sigma] \Rightarrow C[P_2\sigma]} \qquad (TSem)$$

Using weakening and substitution properties proved in [2], we can provide the result that well-formed bases guarantees type preservation.

Proposition 1. *For all $\sigma \in \Sigma_\Delta$, $\emptyset \vdash P\sigma : (L, Pr, U)$ iff $\Delta \vdash P : (L, Pr, U)$.*

Starting from a correct term, all the terms obtained via Δ-(L, Pr, U) safe rules are correct, and thus avoiding conditions over P_1 as they do not influence the type of the obtained term. As desired, typed reduction preserves correctness.

Theorem 1 (Subject Reduction). *If $\emptyset \vdash st : (L, Pr, U)$ and $st \Rightarrow st'$, then $\emptyset \vdash st' : (L', Pr', U')$ for a well-formed type (L', Pr', U').*

4 Type Inference

To formalize type reconstruction, we will need a concise way of talking about the possible ways that type variables can be substituted by types, in a term and its associated context, to obtain a valid typing statement.

In this section we define inference rules in order to derive which rules are Δ-(L, Pr, U) safe, where the choices of Δ, L, Pr, U are guided by the initial pattern. Formally, given a typable pattern P, there exists a typing $\Delta \vdash P : (L, Pr, U)$. The typed semantics of rule $(Tsem)$ does not show how to choose Δ and Pr (L and U depend on Pr).

We assume that for each object variable x there is an o-type variable ρ_x ranging over basic types, and for each term variable X there is an m-type variable λ_X ranging over multisets of basic types. Moreover, we assume that Λ ranges over unions of multisets of basic types, o-type and m-type variables.

A basic scheme Θ is a mapping from atomic variables to their o-type variables, and from term variables to triples of their l-type variables, pr-type variables and u-type variables:

$$\Theta ::= \emptyset \mid \Theta, x : \rho_x \mid \Theta, X : \lambda_X.$$

The rules for inferring the typings use judgements of the form:

$$\vdash P : \Theta; (L_\Lambda, \Lambda, U_\Lambda); \Xi$$

where Θ is the basis scheme in which P is well-formed, $(L_\Lambda, \Lambda, U_\Lambda)$ is the type of P, and Ξ is the set of constraints that should be satisfied. Table 2 presents the inference rules, derived from the typing rules of Table 1.

Example 7. Let us assume a set of basic types $T = \{t_a, t_b\}$, a classification $\Gamma = \{a : t_a, b : t_b\}$ and the functions min, max defined as:

$min(t_1, t_2)$

$t_1 \backslash t_2$	t_1	t_2
t_1	$\diamond$	0.6
t_2	0.25	$\diamond$

$max(t_1, t_2)$

$t_1 \backslash t_2$	t_1	t_2
t_1	$\diamond$	4
t_2	5/3	$\diamond$

Consider the well-formed term $a^5\ b^2$. In order to infer its type there are several ways of considering the patterns P_1 and P_2 in order to apply the last $(IPar)$ rule. Consider the following two cases:

- $P_1 = a^3b^2$ (well-formed) and $P_2 = a^2$ (well-formed); in this case the rule $(IPar)$ leads to a well-formed type with all constraints satisfied;
- $P_1 = ab^2$ (not well-formed) and $P_2 = a^4$ (well-formed); in this case the type-compatibility condition is not fulfilled even if the obtained term is well-formed.

In order to obtain the type of the term $a^5\ b^2$, there are considered different decompositions until there is obtained an well-formed type and all conditions hold, or all possible decompositions do not provide a well-formed type with the conditions fulfilled.

Table 2. Inference Rules

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset, \emptyset); \emptyset \quad (IEps) \qquad \frac{a : t \in \Gamma}{\vdash a : \emptyset; (L_t, t, U_t); \emptyset} \quad (IObj)$$

$$\vdash x : \{x : \rho_x\}; (L_{\rho_x}, \rho_x, U_{\rho_x}); \emptyset \qquad (IVar1)$$

$$\vdash X : \{X : \lambda_X\}; (L_{\lambda_X}, \lambda_X, U_{\lambda_X}); \emptyset \qquad (IVar2)$$

$$\frac{\vdash v : \Theta; (L_\Lambda, \Lambda, U_\Lambda); \Xi \quad \vdash P' : \Theta'; (L_{\Lambda'}, \Lambda', U_{\Lambda'}); \Xi' \quad i \in Lab}{\vdash [P']^i_v : \Theta \cup \Theta'; (L_\Lambda, \Lambda, U_\Lambda); \Xi \cup \Xi' \cup \{(L_\Lambda, \Lambda, U_\Lambda) \bowtie (L_{\Lambda'}, \Lambda', U_{\Lambda'})\}} \quad (IMem)$$

$$\frac{\vdash P : \Theta; (L_\Lambda, \Lambda, U_\Lambda); \Xi \quad \vdash P' : \Theta'; (L_{\Lambda'}, \Lambda', U_{\Lambda'}); \Xi'}{\begin{array}{c}\vdash PP' : \Theta \cup \Theta'; (L_{\Lambda+\Lambda'}, \Lambda + \Lambda', U_{\Lambda+\Lambda'}); \\ \Xi \cup \Xi' \cup \{(L_\Lambda, \Lambda, U_\Lambda) \bowtie (L_{\Lambda'}, \Lambda', U_{\Lambda'})\}\end{array}} \quad (IPar)$$

The rules of Table 2 are easily derived from the rules of Table 1. The basis is the union of the basis of the composing patterns, without renaming, because each variable x or X is associated with an unique o-type variable, or to an unique m-type variable, respectively. The key difference between inference rules of Table 2 and typing rules of Table 1 is that the conditions of type compatibility and type satisfaction are not premises, but conclusions. In this way, at the end of inference, all these conditions create a set of constraints that must be checked to decide the applicability of the rules.

Soundness and completeness of our inference rules can be stated as usual. A type mapping associates o-type variables to basic types, m-type variables to multisets of basic types. A type mapping $\mathbf{m}$ satisfies a set of constraints Ξ if all the constraints in $\mathbf{m}(\Xi)$ hold.

Theorem 2 (Soundness of Type Inference).
If $\vdash P : \Theta; (L_\Lambda, \Lambda, U_\Lambda); \Xi$ *and* $\mathbf{m}$ *is a type mapping satisfying* Ξ*, then*

$$\mathbf{m}(\Theta) \vdash P : (\mathbf{m}(L_\Lambda), \mathbf{m}(\Lambda), \mathbf{m}(U_\Lambda)).$$

Theorem 3 (Completeness of Type Inference). *If* $\Delta \vdash P : (L_{Pr}, Pr, U_{Pr})$*, then* $\vdash P : \Theta; (L_\Lambda, \Lambda, U_\Lambda); \Xi$ *for some* Θ*,* Λ*,* Ξ *such that there is a type mapping* $\mathbf{m}$ *that satisfies* Ξ*,* $\Delta \supseteq \mathbf{m}(\Theta)$*, and* $Pr = \mathbf{m}(\Lambda)$*.*

We use inference rules to decide applicability of typed reduction rules for Δ-(L_{Pr}, Pr, U_{Pr}) safe rules. The first step is to see when a type mapping ensures that a rule is a Δ-(L_{Pr}, Pr, U_{Pr}) safe rule, i.e. when it satisfies the constraints of Definition 7. The concept of Δ-(L_{Pr}, Pr, U_{Pr}) safety is used to classify rules according to the types we can derive for the right hand side patterns of them.

Lemma 1 (Characterisation of Δ-(L, Pr, U) safe rules).
A rule $P_1 \to P_2$ *is* Δ-(L_{Pr}, Pr, U_{Pr}) *safe if and only if the type mapping* $\mathbf{m}$ *defined by the basis* Δ*, i.e. such that*

- $\mathbf{m}(\rho_x) = t$ *if* $\Delta(x) = t$*, and*
- $\mathbf{m}(\lambda_X) = Pr$ *if* $\Delta(X) = (L_{Pr}, Pr, U_{Pr})$*.*

satisfies the set of constraints $\Xi_2 \cup \{\Lambda_2 = Pr\}$*, whenever it holds that* $\vdash P_2 : \Theta_2; (L_{\Lambda_2}, \Lambda_2, \Omega_2); U_{\Lambda_2}$*.*

Since Δ-(L_{Pr}, Pr, U_{Pr}) safe rules can be applied only in contexts in which type (L_{Pr}, Pr, U_{Pr}) fits, we must characterise also the fitting relation.

Lemma 2 (Characterisation of Fitting Relation).
Let the context C *be such that* $\vdash C[T] : (L_{Pr_C}, Pr_C, U_{Pr_C})$ *for some* T *and well-formed type* $(L_{Pr_C}, Pr_C, U_{Pr_C})$*. A well-formed type* (L_{Pr}, Pr, U_{Pr}) *fits* C *if and only if the type mapping* $\mathbf{m}$ *defined by*

$$\mathbf{m}(\lambda_X) = Pr$$

satisfies the set of constraints

$$\Xi_C \cup \{(L_{\Lambda_C}, \Lambda_C, U_{\Lambda_C}) \text{ is well-formed}\},$$

where $\vdash C[X] : \{X : \lambda_X\}; (L_{\Lambda_C}, \Lambda_C, U_{\Lambda_C}); \Xi_C$*.*

Using the characterisation of Δ-(L, Pr, U) safe rules and the fitting relation, we can infer the applicability of a rewriting rule by checking if the type mapping respects the required constraints.

Theorem 4 (Applicability of Rewriting Rules).
If the following conditions are fulfilled
$\vdash P_1 : \Theta_1; (L_{\Lambda_1}, \Lambda_1, U_{\Lambda_1}); \Xi_1$*,*
$\vdash P_2 : \Theta_2; (L_{\Lambda_2}, \Lambda_2, U_{\Lambda_2}); \Xi_2$*,*
$\vdash C[X] : \{X : \lambda_X\}; (L_{\Lambda_C}, \Lambda_C, U_{\Lambda_C}); \Xi_C$
and $P_1\sigma \neq \epsilon$*, then the rule* $P_1 \to P_2$ *can be applied to the term* $C[P_1\sigma]$ *such that* $\vdash C[P_1\sigma] : (L_{Pr_C}, Pr_C, U_{Pr_C})$ *for a well-formed type* $(L_{Pr_C}, Pr_C, U_{Pr_C})$ *if and only if the type mapping* $\mathbf{m}$ *defined by*

- $\mathbf{m}(\rho_x) = t$, *if* $\sigma(x) : t \in \Gamma$,
- $\mathbf{m}(\lambda_X) = Pr$, *if* $\vdash \sigma(X) : (L_{Pr}, Pr, U_{Pr})$

satisfies the set of constraints

$$\Xi_2 \cup \{(\lambda_X = \Lambda_2)\} \cup \Xi_C \cup \{(L_{\Lambda_C}, \Lambda_C, U_{\Lambda_C}) \text{ is well-formed}\}.$$

5 Conclusion

This paper is related to a previous approach presented in [2], where a quantitative types based on ratio thresholds is introduced. The inspiration was the first discovered ion transporter (awarded with a Nobel Prize in 1997), namely the sodium-potassium pump, which extrudes sodium ions in exchange for potassium ions. These exchanges take place only if the ratios of these elements are between certain lower and upper thresholds. To cope properly with such constraints, we introduced in a multiset-based type system with ratio thresholds, where the sodium/potassium pump is used as a running example. We associated to each system a set of constraints, and relate them to the ratios between elements. If the constraints are satisfied, we prove that if a system is well-typed and an evolution rule is applied, then the obtained system is also well-typed.

In this paper we defined a type inference algorithm for membrane systems with integral proteins for which soundness and completeness are proved. The work intends to allow automatic analyse of inference according to the defined type system, and to provide support for the development of software tools.

The proposed typed semantics completely excludes the fact that sometimes biological constraints can be broken leading to a disease or even to the death of the biological system. However, the typed semantics can be modified in order to allow transitions that lead to terms that are not typable. In this case the type system should signal that some undesired event has been reached. In this way, it can be checked if a term breaks some biological property, or if the system has some unwanted behaviour.

Acknowledgements. The work was supported by a grant of the Romanian National Authority for Scientific Research, project number PN-II-ID-PCE-2011-3-0919.

References

1. Aman, B. and Ciobanu, G. Typed Membrane Systems. *Lecture Notes in Computer Science* **5957**, 169–181 (2010).
2. Aman, B. and Ciobanu, G. Behavioural Types Inspired by Cellular Thresholds. *Lecture Notes in Computer Science* **8368**, 1–15 (2014).
3. Besozzi, D. and Ciobanu, G. A P System Description of the Sodium-Potassium Pump. *Lecture Notes in Computer Science* **3365**, 210–223 (2005).
4. Bioglio, L. Enumerated Type Semantics for the Calculus of Looping Sequences. *RAIRO - Theoretical Informatics and Applications* **45** (1), 35–58 (2011).

5. Cavaliere, M. and Sedwards, S. Modelling Cellular Processes Using Membrane Systems With Peripheral and Integral Proteins. *Lecture Notes in Bio-Informatics* **4210**, 108–126 (2006).
6. Ciobanu, G., Păun, Gh. and Pérez-Jiménez, M.J.(Eds.) *Applications of Membrane Computing*. Springer (2006).
7. Ciobanu, G. *Membrane Computing and Biologically Inspired Process Calculi*. "A.I.Cuza" University Press, Iaşi (2010).
8. Hsu, S.-B., Hwang, T.-W. and Kuang, Y. A Ratio-Dependent Food Chain Model and Its Applications to Biological Control. *Mathematical Biosciences* **181**, 55–83 (2003).
9. Păun, Gh. *Membrane Computing. An Introduction*. Springer (2002).
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Handbook of Membrane Computing*. Oxford University Press (2010).
11. Salomaa, A. *Formal Languages*. Academic Press (1973).
12. Wells, J. The Essence of Principal Typings. *Lecture Notes in Computer Science* **2380**, 913–925 (2002).

Solving SAT with Active Membranes and Pre-Computed Initial Configurations

Bogdan Aman, Gabriel Ciobanu

Romanian Academy, Institute of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Summary. In this paper we provide algorithms for solving the SAT problem using P systems with active membranes with neither polarization nor division rules. The semi-uniform solutions are given under the assumption that initial configurations (either alphabet or structure) of exponential size are pre-computed by well-defined P systems (P systems with replicated rewriting and P systems with active membranes and membrane creation, respectively) working in polynomial time. An important observation is that we specify how the pre-computed initial configurations are constructed.

1 Introduction

Membrane computing is inspired by the architecture and the behaviour of living cells. Various classes of membrane systems (also called P systems) have been defined in [9], while several applications of these systems are described in [3]. Membrane systems are characterised by three features: (i) a membrane structure consisting of a hierarchy of membranes (which are either disjoint or nested), with an unique top membrane called the *skin*; (ii) multisets of objects associated with membranes; (iii) rules for processing the objects and membranes. When membrane systems are seen as computing devices, two main research directions are usually considered: computational power in terms of the classical notion of Turing computability (e.g., [1]), and efficiency in algorithmically solving NP-complete problems in polynomial time (e.g., [2]). Thus, membrane systems define classes of computing devices which are both powerful and efficient.

Under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency of the system ways to exponentially grow the workspace during the computation, nondeterminism, and so on). For instance, some pre-computed resources are used in [4, 6].

In this paper we consider P systems with active membranes [7], and show that they can provide simple semi-uniform solutions to the SAT problem without using neither polarization nor division, but using exponential size pre-computed

initial configurations (either alphabet or structure). An important observation is that we specify how our pre-computed initial configurations are constructed in a polynomial number of steps by additional well-defined P systems (P systems with replicated rewriting and P systems with active membranes and membrane creation, respectively).

2 Preliminaries

We consider polarizationless P systems with active membranes [7]. The original definition also includes division rules, rules that are not needed here.

Definition 1. *A* polarizationless P system with active membranes *is a tuple* $\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$, *where:*

- $d \geq 1$ *is the* initial degree*;*
- Γ *is a finite non-empty alphabet of* objects*;*
- Λ *is a finite* set of labels *for membranes;*
- μ *is a* membrane structure *(i.e., a rooted unordered tree, usually represented by nested brackets) in which each membrane is labelled by an element of* Λ *in a one-to-one way;*
- $w_1, \ldots, w_d$ *are strings over* Γ*, describing the* initial multisets *of objects placed in a number of* d *membranes of* μ*;*
- R *is a finite* set of rules *over* Γ*:*
  1. $[a \to w]_h$ object evolution rules
     An object a *is rewritten into the multiset* w*, if* a *is placed inside a membrane labelled by* h*.*
  2. $a[\ ]_h \to [b]_h$ send-in communication rules
     An object a *is sent into a membrane labelled by* h*, becoming* b*.*
  3. $[a]_h \to b[\ ]_h$ send-out communication rules
     An object a*, placed into a membrane labelled by* h*, is sent out of membrane* h *and becomes* b*.*
  4. $[a]_h \to b$ dissolution rules
     A membrane h *containing an object* a *is disslved, while object* a *is rewritten to* b*.*

Each configuration $\mathcal{C}_i$ of a P system with active membranes and input objects is described by the membrane structure, together with the multisets of objects located in the corresponding membranes. The initial configuration of such a system is denoted by $\mathcal{C}_0$. An evolution step $\mathcal{C}_i \Rightarrow \mathcal{C}_{i+1}$ from a configuration $\mathcal{C}_i$ to a new configuration $\mathcal{C}_{i+1}$ is done according to the following principles:

- Each object is involved in at most one rule per step, while each membrane could be involved in several rules.
- The application of rules is maximally parallel: all rules that can be applied are applied.

- When several conflicting rules could be applied at the same time, a nondeterministic choice is performed; this implies that multiple configurations can be reached as the result of an evolution step.
- In each evolution step, all evolution rules are applied inside the most inner membranes, followed by all communication rules involving the membranes themselves. This process is then repeated to the membranes containing them, and so on towards the skin membrane.
- Objects sent out from the skin membrane represent the computation result.

A *halting evolution* of such a system Π is a finite sequence of configurations $\overrightarrow{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, such that $\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \ldots \Rightarrow \mathcal{C}_k$, and no rules can be applied any more in $\mathcal{C}_k$. A *non-halting* evolution $\overrightarrow{\mathcal{C}} = (\mathcal{C}_i \mid i \in \mathbb{N})$ consists of infinite evolution $\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \ldots$, where the applicable rules are never exhausted.

3 Solving the SAT Problem with Active Membranes

At the beginning of 2005, Gh. Păun wrote:

"My favourite question (related to complexity aspects in P systems with active membranes and with electrical charges) is that about the number of polarizations. Can the polarizations be completely avoided? The feeling is that this is not possible - and such a result would be rather sound: passing from no polarization to two polarizations amounts to passing from non-efficiency to efficiency."

This conjecture (problem F in [8]) can be formally described in terms of membrane computing complexity classes as follows:

$$P = PMC_{\mathcal{AM}^0(+d,-n,+e,+c)}$$

where

- $PMC_{\mathcal{R}}$ indicates that the result holds for P systems with input membrane;
- $+d$ indicates that dissolution rules are permitted;
- $-n$ indicates that only division rules for elementary membranes are allowed;
- $+e$ indicates that evolution rules are permitted;
- $+c$ indicates that communication rules are permitted.

The SAT problem checks the satisfiability of a propositional logic formula in conjunctive normal form (CNF). Let $\{x_1, x_2, \ldots, x_n\}$ be a set of propositional variables. A formula in CNF is of the form $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each C_i, $1 \leq i \leq m$ is a disjunction of the form $C_i = y_1 \vee y_2 \vee \cdots \vee y_r$ $(r \leq n)$, where each y_j is either a variable x_k or its negation $\neg x_k$.

We present some attempts to solve this conjecture by providing algorithms solving the SAT problem using P systems with active membranes with neither polarizations nor division, but using exponential pre-computed initial configurations constructed by additional P systems in polynomial time.

3.1 Solving SAT Problem by Using a Pre-Computed Alphabet

In this section, we propose a polynomial time solution to the SAT problem using the polarizationless P systems with active membranes, without division, but with a pre-computed alphabet. For any instance of SAT we construct effectively a system of membranes that solves it. Formally, we prove the following result:

Theorem 1. *The SAT problem can be solved by a* **polarizationless** *P system with active membranes and* **without division**, *but with an exponential alphabet pre-computed in linear time with respect to the number of variables and the number of clauses, i.e.,*

$$P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\alpha))} \cdot$$

where $pre(\alpha)$ *indicates that a pre-computed alphabet is permitted.*

Proof. Let us consider a propositional formula

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

where each $C_i, 1 \leq i \leq m$ is a disjunction of the form

$$C_i = y_1 \vee y_2 \vee \cdots \vee y_r \ (r \leq n),$$

where each y_j is either a variable x_k or its negation $\neg x_k$.

We construct a P system with active membranes able to check the satisfiability of φ. The P system is given by $\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$, where:

- $V = \{z_i \mid 0 \leq i \leq max\{m, n\}\} \cup$
  $\cup \{s_i \mid i = t_1 \ldots t_n, t_j \in \{0,1\} \text{ and } 1 \leq j \leq n\} \cup \{yes, no\}$.
  The alphabet $\{s_i \mid i = t_1 \ldots t_n, t_j \in \{0,1\} \text{ and } 1 \leq j \leq n\}$ to be placed inside the input membrane 0 can be generated, starting from an object s, using the rules:
  - $s \to s_0 s_1$;
  - $s_i \to s_{i0} s_{i1}$, for $i = t_1 \ldots t_k$ where $t_j \in \{0,1\}$ and $1 \leq j \leq k < n$.

  Thus all the possible assignments for the n variable $\{x_1, x_2, \ldots, x_n\}$ are created. The rules are applied until the length k of i in the second rule equals n. For example, s_{100} over $\{x_1, x_2, x_3\}$ represents the assignment $x_1 = 1$, $x_2 = 0$ and $x_3 = 0$ (1 stands for $true$, while 0 stands for $false$). The input alphabet can be computed in linear (polynomial) time by using an additional device, for instance P systems with replicated rewriting [5].
- $\Lambda = \{0, c_1, \ldots, c_m, h\}$, with $c_i = z_1 \ldots z_n$, $1 \leq i \leq m$ where
  - $z_j = 1$ if x_j appears in C_i;
  - $z_j = 0$ if $\neg x_j$ appears in C_i;
  - $z_j = \star$ if neither x_j nor $\neg x_j$ appear in C_i.

  For example $c_1 = 1 \star 0$ over the set of variables $\{x_1, x_2, x_3\}$ represents the disjunction $c_1 = x_1 \vee \neg x_3$.
- $\mu = [[[\ldots[[[\ ]_0]_{c_1}]_{c_2} \ldots]_{c_{m-1}}]_{c_m}]_h$.
- $w_0 = z_0$.
- $w_i = \lambda$, for all $i \in \Lambda \backslash \{0\}$.
- The set R contains the following rules:

1. $[z_0]_0 \rightarrow z_0$
   After the input is placed inside membrane 0, membrane 0 is dissolved, and its content is released in the upper membrane labelled with c_1.
2. $[s_i]_{c_j} \rightarrow s_i[\ ]_{c_j}$
   if i and j have at least one position with the same value (either 0 or 1);
   $[s_i]_{c_m} \rightarrow yes$
   if i and m have at least one position with the same value (either 0 or 1).

   An assignment s_i is sent out of a membrane c_m if there is at least one position in i and j that is equal, namely an assignment to a variable x_k such that it makes C_j true. Once an object *yes* is generated, another object *yes* cannot be created because membrane c_m was dissolved and the rule $[s_i]_{c_m} \rightarrow yes$ cannot be applied. For example, if $c_1 = 1 \star 0$ and s_{101} (as described above), then this means that s_{101} satisfies the clause coded by $c_1 = 1 \star 0$ since both have 1 on their first position, and this is enough to make true a disjunction.
3. $[z_0 \rightarrow z_1]_{c_1}$
   $[z_i]_{c_i} \rightarrow [\ ]_{c_i} z_{i+1}$, for $1 \leq i \leq m-1$
   $[z_m]_{c_m} \rightarrow no$
   The object z_0 waits a step after membrane 0 is dissolved in order to allow the other objects s_i to go through the c_j membranes. The object z_i then is communicated through the c_j membranes. Once z_m reached the membrane c_m, if membrane c_m still exists (i.e., the rule $[s_i]_{c_m} \rightarrow yes$ was not applied), then the answer *no* is generated. Once an object *yes* or *no* is generated, other objects *yes* or *no* cannot be created because membrane c_m was dissolved, and neither rule $[s_i]_{c_m} \rightarrow yes$ nor $[z_m]_{c_m} \rightarrow no$ can be applied.
4. $[yes]_h \rightarrow yes[\ ]_h$
   $[no]_h \rightarrow no[\ ]_h$
   The answer *yes* or *no* regarding the satisfiability is sent out of the skin.

3.2 Solving SAT Problem Using a Pre-Computed Initial Structure

In this section, we propose a polynomial time solution to the SAT problem using the polarizationless P systems with active membranes and without division, but with a pre-computed structure. For any instance of SAT we construct effectively a system of membranes that solves it. We also enforce another principle needed to perform an evolution step: each membrane can be subject to at most one communication rule per step. This principle is needed when generating all possible assignments to be verified. Formally, we prove the following result:

Theorem 2. *The SAT problem can be solved by a* **polarizationless** *P system with active membranes and* **without division**, *but with an initial exponential structure pre-computed in linear time with respect to the number of variables and the number of clauses, i.e.,*

$$P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\mu))} \cdot$$

where $pre(\mu)$ indicates that a pre-computed structure is permitted.

Proof. Let us consider a propositional formula

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

where each $C_i, 1 \leq i \leq m$ is a disjunction of the form

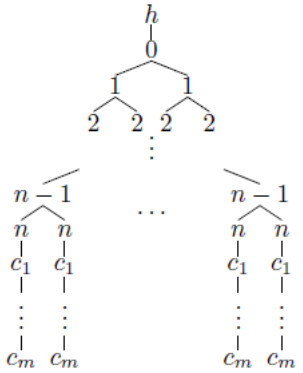
$$C_i = y_1 \vee y_2 \vee \cdots \vee y_r \ (r \leq n),$$

where each y_j is either a variable x_k or its negation $\neg x_k$.

We construct a P system with active membranes able to check the satisfiability of φ. The P system is given by $\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$, where:

- $V = \{a_i, t_i, t'_i, f_i, f'_i \mid 1 \leq i \leq n\} \cup \{z_i \mid 0 \leq i \leq 4 \times n + 2 \times m\} \cup \{yes, no\}$.
- $\Lambda = \{0, \ldots, n, c_1, \ldots, c_m, h\}$, $1 \leq i \leq m$.
- $\mu = [[[[\ldots]_2[\ldots]_2]_1[[\ldots]_2[\ldots]_2]_1]_0]_h$, where
  - each membrane i contains two membranes $i+1$ for $0 \leq i \leq n-1$;
  - each membrane n contains a membrane structure $[[\ldots[\ ]_{c_m}\ldots]_{c_1}]_{c_0}$;
  - membrane 0 is the input membrane.

  Graphically, the membrane structure μ can be represented as a tree:



  This membrane structure can be generate in linear (polynomial) time with respect to the number of variables and the number of clauses. This is done by using an additional device that starts from a membrane structure $[[\ ]_0]_h$, with object 0 placed inside membrane 0 and rules of the form:
  - $[i \to (i+1)'\ (i+1)']_i$, for $0 \leq i \leq n-1$
  - $i' \to [i]_i$, for $1 \leq i \leq n$
  - $n \to [c_2]_{c_1}$
  - $c_k \to [c_{k+1}]_{c_k}$, for $2 \leq k \leq m-1$
  - $c_m \to [\ ]_{c_m}$.
- $w_0 = a_1 z_0$.
- $w_i = \lambda$, for all $i \in \Lambda \backslash \{0\}$.
- The set R contains the following rules:
  1. $[z_i \to z_{i+1}]_0$, for all $0 \leq i < 4 \times n + 2 \times m$
     These rules count the time needed for producing the truth assignments for the n variables inside the membranes labelled by n ($3 \times n$ steps), then to

dissolve the membranes labelled by c_j, $1 \leq j \leq m$ ($2 \times m$ steps), and for an y object to reach the membrane labelled by 0 (n steps).

2. $[a_i \rightarrow t_i f_i]_{i-1}$, for $1 \leq i \leq n$
   $t_i[\ ]_i \rightarrow [t_i]_i$, for $1 \leq i \leq n$
   $f_i[\ ]_i \rightarrow [f_i]_i$, for $1 \leq i \leq n$
   $[t_i \rightarrow t'_i t'_i a_{i+1}]_i$, for $1 \leq i \leq n-1$
   $t'_i[\ ]_k \rightarrow [t_i]_k$, for $i+1 \leq k \leq n$
   $[t_i \rightarrow t'_i t'_i]_k$, for $i+1 \leq k \leq n-1$
   $[f_i \rightarrow f'_i f'_i a_{i+1}]_i$, for $1 \leq i \leq n-1$
   $f'_i[\ ]_k \rightarrow [f_i]_k$, for $i+1 \leq k \leq n$
   $[f_i \rightarrow f'_i f'_i]_k$, for $i+1 \leq k \leq n-1$
   In membranes n we create all possible assignments for the n variable $\{x_1, x_2, \ldots, x_n\}$. It starts from an object a_1 placed initially in membrane labelled by 0. Each a_i is used to create t_i and f_i that are then send in one of the two membranes labelled by i placed in membrane $i-1$. In fact each membrane i receives either t_i or f_i, and this is possible because a membrane can be involved in only one communication rule of an evolution step. After an object t_i or f_i reaches a membrane i, it generates two new copies of it to be sent inside membranes $i+1$ together with an object a_{i+1} that is used then to construct the assignments over variable x_{i+1}.
3. $t_i[\ ]_{c_j} \rightarrow [t_i]_{c_j}$, if x_i appears in C_j
   $[t_i]_{c_j} \rightarrow t_i$, for $1 \leq i \leq n$, $1 \leq j < m$
   $[t_i]_{c_m} \rightarrow y$, for $1 \leq i \leq n$
   $f_i[\ ]_{c_j} \rightarrow [t_i]_{c_j}$, if $\neg x_i$ appears in C_j
   $[f_i]_{c_j} \rightarrow f_i$, for $1 \leq i \leq n$, $1 \leq j \leq m$
   $[f_i]_{c_m} \rightarrow y$, for $1 \leq i \leq n$.

   An assignment t_i (f_i) is sent into a membrane c_j if there is an assignment to a variable x_k ($\neg x_k$) such that it makes C_j true. Once all membranes labelled by c_i are dissolved inside a membrane labelled by n, an object y is generated.
4. $[y]_k \rightarrow [\ ]_k y$, for $k \in \Lambda \backslash \{0, h\}$
   $[y]_0 \rightarrow yes$
   $[z_{4\times n+2\times m}]_0 \rightarrow no$.
   The object z_0 waits for $4 \times n + 2 \times m$ steps in order to allow dissolving the membrane labelled by 0 if this still exists (i.e., the rule $[y]_0 \rightarrow yes$ was not applied), then the answer no is generated. Once an object yes or no is generated, other objects yes or no cannot be created because membrane c_m was dissolved, and neither rule $[y]_0 \rightarrow yes$ nor $[z_{4\times n+2\times m}]_0 \rightarrow no$ can be applied.
5. $[yes]_h \rightarrow yes[\ ]_h$
   $[no]_h \rightarrow no[\ ]_h$.
   The answer yes or no regarding the satisfiability is sent out of the skin.

Example 1. We illustrate this algorithm and the evolution of a system Π constructed for the propositional formula $\psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$.

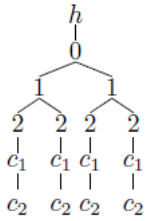
Thus, m=n=2. The initial configuration of the systems, constructed by an additional device that starts from a membrane structure $[[\ ]_0]_h$, with object 0 placed inside membrane 0 and rules of the form:

- $[0 \to 1'\ 1']_0$ and $[1 \to 2'\ 2']_1$
- $1' \to [1]_1$ and $2' \to [2]_2$
- $2 \to [c_2]_{c_1}$ and $c_2 \to [\ ]_{c_2}$.

The obtained structure is

$$[[[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1 a_1 z_0]_0]_h$$

Graphically, the membrane structure μ can be represented as a tree:



Using the set R of rules $1 \div 5$, the computation proceeds as follows:

$[[[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1 a_1 z_0]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1 t_1 f_1 z_1]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2 t_1]_1[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2 f_1]_1 z_2]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2 t_1' t_1' a_2]_1[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2 f_1' f_1' a_2]_1 z_3]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2}]_{c_1} t_1]_2[[[\ ]_{c_2}]_{c_1} t_1]_2 t_2 f_2]_1[[[[\ ]_{c_2}]_{c_1} f_1]_2[[[\ ]_{c_2}]_{c_1} f_1]_2 t_2 f_2]_1 z_4]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2}]_{c_1} t_1 t_2]_2[[[\ ]_{c_2}]_{c_1} t_1 f_2]_2]_1[[[[\ ]_{c_2}]_{c_1} f_1 t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_5]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2} t_1]_{c_1} t_2]_2[[[\ ]_{c_2} t_1]_{c_1} f_2]_2]_1[[[[\ ]_{c_2} t_2]_{c_1} f_1]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_6]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2} t_1 t_2]_2[[\ ]_{c_2} t_1 f_2]_2]_1[[[\ ]_{c_2} t_2 f_1]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_7]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2} t_1 t_2]_2[[f_2]_{c_2} t_1]_2]_1[[[f_1]_{c_2} t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_8]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2} t_1 t_2]_2[y t_1]_2]_1[[y t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_9]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2} t_1 t_2]_2[t_1]_2 y]_1[[t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2 y]_1 z_{10}]_0]_h$
$\Rightarrow [[[[[\ ]_{c_2} t_1 t_2]_2[t_1]_2]_1[[t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 y y z_{11}]_0]_h$
$\Rightarrow [[[[\ ]_{c_2} t_1 t_2]_2[t_1]_2]_1[[t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 y z_{12} yes]_h$
$\Rightarrow [[[[\ ]_{c_2} t_1 t_2]_2[t_1]_2]_1[[t_2]_2[[[\ ]_{c_2}]_{c_1} f_1 f_2]_2]_1 y z_{12}]_h yes$

It can be noticed that even the object z has now the subscript $4 \times n + 2 \times m = 4 \times 2 + 2 \times 2 = 12$, it cannot generate a *no* object because membrane labelled by 0 was already dissolved by an y object in the previous step. Also, even another y object reached the membrane labelled by 0, it cannot generate an *yes* object because membrane labelled by 0 was already dissolved by another y object in a previous step.

4 Conclusion

In this paper we deal with a question presented by Păun in 2005: *Can the polarizations be completely avoided?* (related to complexity aspects of P systems with active membranes and with electrical charges). We answer positively to this question: we do not use polarizations in solving the SAT problem, but use a pre-computed initial configuration involving either exponential alphabet or exponential structure.

We proved $P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\alpha))}$ and $P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\mu))}$ by providing two algorithms for solving the SAT problem using **polarizationless** P system with active membranes and **without division**. For the former equality, the provided algorithm is using an exponential alphabet pre-computed in linear time by a P system with replicated rewriting, while the later one is using an initial exponential structure pre-computed in linear time with respect to the number of variables and the number of clauses by P systems with membrane creation.

References

1. B. Aman, G.Ciobanu. Turing Completeness Using Three Mobile Membranes. *Lecture Notes in Computer Science* **5715**, 42–55 (2009).
2. B. Aman, G. Ciobanu. *Mobility in Process Calculi and Natural Computing.* Natural Computing Series, Springer (2011).
3. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.). *Applications of Membrane Computing.* Springer (2006).
4. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang. Deterministic Solutions to QSAT and Q3SAT by Spiking Neural P Systems with Pre-Computed Resources. *Theoretical Computer Science* **411**(25), 2345–2358 (2010).
5. S.N. Krishna, R. Rama. P Systems with Replicated Rewriting. *Journal of Automata, Languages and Combinatorics* **6**(3), 345–350 (2001).
6. A. Leporati, M.A. Gutiérrez-Naranjo. Solving Subset Sum by Spiking Neural P Systems With Pre-Computed Resources. *Fundamenta Informaticae* **87**(1), 61–77 (2008).
7. Gh. Păun. P Systems With Active Membranes: Attacking NP-complete Problems. *Journal of Automata, Languages and Combinatorics* **6**, 75–90 (2001).
8. Gh. Păun. Further Twenty Six Open Problems in Membrane Computing. *Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez et al. eds.), Fénix Editora, Sevilla, 249–262 (2005).
9. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.). *The Oxford Handbook of Membrane Computing*, Oxford University Press (2010).

Red-Green P Automata

Bogdan Aman[1], Erzsébet Csuhaj-Varjú[2], Rudolf Freund[3]

[1] Institute of Computer Science, Romanian Academy
Iaşi, Romania, Email: bogdan.aman@gmail.com
[2] Faculty of Informatics, Etvs Loránd University
Budapest, Hungary, Email: csuhaj@inf.elte.hu
[3] Faculty of Informatics, Vienna University of Technology
Vienna, Austria, Email: rudi@emcc.at

Summary. In this short note we extend the notion of red-green Turing machines to specific variants of P automata. Acceptance and recognizability of finite strings by red-green automata are defined via infinite runs of the automaton on the input string and the way how to distinguish between red and green states.

1 Introduction

In this short note we introduce the notion of red-green automata in the area of P systems. Acceptance and recognizability of finite strings by a red-green Turing machine are defined via infinite runs of the automaton on the input string and the way how to distinguish between red and green states; via infinite runs which are allowed to change between red and green states more than once, more than the recursively enumerable sets of strings can be obtained, i.e.,in that way we can "go beyond Turing". Various possibilities how to "go beyond Turing" to be already found in the literature are discussed in [9]; most of the definitions and results for red-green Turing machines are taken from this paper. In the area of P systems, first attempts to do that can be found in [5] and [8]. Computations with infinite words by P automata have been investigated in [4].

Here we focus on the idea of being able to switch between red and green states in P automata, where states are specific properties of a configuration, for example, the occurrence or the non-occurrence of a specific symbol. As for Turing machines, with one change from red to green states, we can accept all recursively enumerable languages. A similar result can easily be obtained for many variants of P automata, especially for the basic model using antiport rules assigned to the skin membrane.

In this note we only focus on the concept of red-green automata for P automata, without giving formal definitions or proofs, as we assume the reader to know the underlying notions and concepts from formal language theory (e.g., see) as well as from the area of P systems (e.g., see). A lot of research topics wait for being

investigated for P automata "going beyond Turing", but as well for the idea of having red and green configurations together with models of P automata which are not computationally complete, as for example dP automata.

2 Red–Green Turing Machines

A Turing machine M is called a *red–green Turing machine* if its set of internal states Q is partitioned into two subsets, Q_r and Q_g, and M operates without halting. Q_r is called the set of red states, Q_g the set of green states.

Red–green Turing machines can be seen as a type of ω-Turing machines on finite inputs with a recognition criterion based on some property of the set(s) of states visited (in)finitely often, in the tradition of ω-automata (see [4]), i.e., we call an infinite run of the Turing machine on input w *recognizing* if and only if

- no red state is visited infinitely often and
- some green states (one or more) are visited infinitely often.

Comment. In the following, "mind change" means changing the color, i.e., changing from red to green or vice versa.

To get the reader familiar with the basic idea of red-green automata, we give a short sketch of the proofs for some well-known results (see [9]):

Theorem 1. *A set of strings L is recognized by a red–green TM with one mind change if and only if $L \in \Sigma_1$, i.e., if L is recursively enumerable.*

Proof. Let L be the set of strings recognized by a red–green TM M with one mind change. Then design a TM that enumerates all possible inputs, simulates and dovetails the computations of M on these inputs, and outputs string w whenever M makes its first mind change (if any) during the computation on w.

Conversely, if $L \in \Sigma_1$ and M is the TM that enumerates L, then design a red–green TM that on input w simulates the computation of M in red but switches to green when w appears in the enumeration. This machine precisely recognizes L. □

2.1 Red-Green Turing Machines – Going Beyond RE

If more mind changes are allowed, the full power of red–green Turing machines is revealed. For example, the complement of a recursively enumerable set L need not be recursively enumerable, too, but it is always red–green recognizable:

Let M' be the TM recognizing L. Then construct a red–green TM M that operates on inputs w as follows: starting in red, the machine immediately switches to green and starts simulating M' on w. If M' halts (thus recognizing w), the machine switches to red and stays in red from then onward. It follows that M precisely recognizes, in fact accepts, the set L. (Acceptance means that for every

word not recognized by the TM it will never make an infinite number of mind changes, i.e., it finally will end up in red.)

The following result characterizes the computational power of red–green Turing machines (see [9]):

Theorem 2. *(i) Red–green Turing machines recognize exactly the Σ_2 sets of the Arithmetical Hierarchy.*

(ii) Red–green Turing machines accept exactly the Δ_2 sets of the Arithmetical Hierarchy.

3 The basic Model of P Automata

The basic model of P automata as introduced in [2] and in a similar way in [3] is based on antiport rules, i.e., on rules of the form u/v, i.e., the multiset u goes out through the membrane and v comes in instead. As it is already folklore, only one membrane is needed for obtaining computational completeness with only one membrane; the input string is defined as the sequence of terminal symbols taken in during a halting computation. Restricting ourselves to P automata with only one membrane as the basic model, we define a P automaton as follows:

A *P automaton* is a construct

$$\Pi = (O, T, w, R)$$

where

- O is the alphabet of objects,
- T is the terminal alphabet,
- w is the multiset of objects present in the skin membrane at the beginning of a computation, and
- R is a finite set of antiport rules.

The strings accepted by Π consist of the sequences of terminal symbols taken in during a halting computation.

Let us cite from [8]:

> "... a super-Turing potential is naturally and inherently present in evolution of living organisms."

In that sense, we now seek for this potential in P automata.

4 Red-Green P Automata

The main challenge is how to define "red" and "green" states in P automata. In fact, states sometimes are considered to simply be the configurations a P automaton may reach during a computation, or some specific elements occurring in a configuration define its state.

Another variant is to consider the multiset applicable to a configuration as its state, which especially makes sense in the case of deterministic systems. Yet then these multisets have to be divided into "red" and "green" ones.

The easiest way to do this is to specify a subset of the rules as green rules, and all multisets consisting of such green rules only constitute the set of all "green" multisets, whereas all the other ones are "red" multisets.

A stronger condition is to divide the set of rules into "red" and "green" and to define the set of "red" and "green" multisets as those which only consist of "red" and "green" rules, respectively. But then the problem arises how to deal with the multisets of rules consisting of rules of both colors.

5 First Results

As is well known, even with the basic model of P automata as defined above we obtain computational completeness by easy simulations of register machines (which themselves are known, even with only two registers, to be able to simulate the actions of a Turing machine). Hence, the following results are direct consequences of the results known for Turing machines:

Theorem 3. *A set of strings L is recognized by a red–green P automaton with one mind change if and only if $L \in \Sigma_1$, i.e., if L is recursively enumerable.*

Theorem 4. *(i) Red–green P automata recognize exactly the Σ_2 sets of the Arithmetical Hierarchy.*

(ii) Red–green P automata accept exactly the Δ_2 sets of the Arithmetical Hierarchy.

Proof. (Sketch) Let TM be a Turing machine and RM be a register machine simulating TM having its set of internal states Q partitioned into two subsets, Q_r and Q_g; TM operates without halting; Q_r is the set of red states, Q_g the set of green states. The register machine can alos colour its states in red and green, but when simulating the actions of TM eventually needs a green and red variant of its states and actions in order to totally stay within the same color as TM when simulating the actions of one computation step of TM. The P automaton $\Pi = (O, T, w, R)$ can simulate the actions of RM very easily, e.g., see Chapter V in [6], without introducing trap symbols, and even in a deterministic way provided

RM is deterministic. The rules in R are of the form qu/pv where q, p are states of RM and u, v are multisets not containing a state symbol. Hence, a configuration can be defined to exactly have the color of the state symbol from RM currently occurring in the skin region. □

One of the main reasons that the proof of the preceding theorems is that easy is based on the fact that the simulation does not need the trick to trap non-wanted evolutions of the system, which is a trick used very often in the area of P systems. Yet this exactly would contradict the basic feature of the red–green automata way of acceptance by looking at *infinite* computations. Fortunately, the basic model of P automata comes along with this nice feature of not needing trap rules for being able to simulate register machines. Only few models of P automata have this nice feature; another variant are P automata with anti-matter, just recentlyintroduced and investigated, see [1].

6 Future Research

Besides investigating the variants of defining "red"/"green", there are various other models of P automata which deserve to be taken into consideration, e.g., dP automata and anti-matter automata.

There are already a lot of strategies and models to be found in the literature how to "go beyond Turing"; some of them should also be of interest to be considered in the P systems area. Thus, a wide range of possible variants to be investigated remains for future research.

References

1. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and anti-matter in membrane systems. *Brainstorming Week in Membrane Computing*, Sevilla, February 2014.
2. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems, in: *Membrane Computing, International Workshop, WMC-CdeA 2002*, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science* **2597**, Springer, 2003, 219–233.
3. R. Freund, M. Oswald: A short note on analysing P systems. *Bulletin of the EATCS* **78**, 2002, 231–236.
4. R. Freund, M. Oswald, L. Staiger: ω-P Automata with Communication Rules. *Workshop on Membrane Computing, 2003*, 203–217, http://dx.doi.org/10.1007/978-3-540-24619-0_15.
5. C.S. Calude, Gh. Păun: Bio-steps beyond Turing. *Biosystems* **77** (2004), 175–194.
6. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
7. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.

8. P. Sosík, O. Valík: On Evolutionary Lineages of Membrane Systems, in: R. Freund et al. (Eds.): *WMC 2005, Lecture Notes in Computer Science* **3850** (2006), 67–78.
9. J. van Leeuwen, J. Wiedermann: Computation as an unbounded process. *Theoretical Computer Science* **429** (2012), 202–212.

Describing Membrane Computations with a Chemical Calculus *

Péter Battyányi, György Vaszil

Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
{battyanyi.peter, vaszil.gyorgy}@inf.unideb.hu

Summary. Membrane systems are nature motivated computational models inspired by certain basic features of biological cells and their membranes. They are examples of the chemical computational paradigm which describes computation in terms of chemical solutions where molecules interact according to rules defining their reaction capabilities. Chemical models can be presented by rewriting systems based on multiset manipulations, and they are usually given as a kind of chemical calculus which might also allow non-deterministic and non-sequential computations. Here we study membrane systems from the point of view of the chemical computing paradigm and show how computations of membrane systems can be described by such a chemical calculus.

1 Introduction

The history of the chemical computing paradigm goes back to the introduction of the Gamma programming language by Bânatre and Le Métayer in [3, 4]. They describe computations in terms of a symbolic chemical solution of molecules with possible reactions between them. The molecules represent the data, the reactions represent their transformations, and the Brownian motion of molecules in the solution represents the execution model of the program.

The general idea behind the chemical paradigm (and the purpose of the introduction of the Gamma formalism) is to be able to express algorithms without the sequentiality which is not inherently necessary, which is not inherently "built into" the problem that the algorithm needs to solve, or in other words, the sequentiality which is the consequence of the given computational model and not related to the logic of the solution of the problem.

The idea was developed further in several different directions, on of them was realized by the chemical abstract machine of Berry and Boudol in [5] where they

* Research supported in part by the Hungarian Scientific Research Fund, "OTKA", grant no. K75952, and by the European Union through the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project which is co-financed by the European Social Fund.

also introduce the notion of membranes which encapsulate subsolutions forcing the reactions to occur in locally isolated ways. The role of membranes was underlined by Păun in [7] where membrane systems (P systems) were introduced emphasizing the importance of the hierarchical structure of different compartments or regions enclosing different molecules which react, evolve according to different rules and they only cross the membranes or region boundaries in a restricted and controlled manner.

Since its introduction, the theory of P systems became an extensive and well established research field, one of the most important and most popular areas of natural computing which also evolved in several different directions and into different subfields. The evolution of the area made its relationship to other chemical computing models and formalisms less apparent, so it might be of interest to examine them from the point of view of the chemical computing paradigm, and to point out the links between P systems and other chemical computational models, as we attempt in the present paper.

This approach could be beneficial in different ways. By being able to translate chemical programs, or chemical computing formalisms (like Gamma) to membrane systems, we could provide something like a high-level programming language for P systems which could serve as an elegant and efficient way of presenting P system algorithms. A preliminary study of turning certain simple Gamma programs to P systems was initiated in [11]. On the other hand, by being able to describe membrane systems using one of the chemical computing formalisms (as we attempt in this paper), we would be able to use the tools and techniques developed for the many different types of chemical calculi to reason about membrane systems and their computations.

In what follows we first give a short introduction to the Gamma formalism and to the notions of membrane systems and their computations. Then we present the γ-calculus from [1], and show how computations of certain membrane systems can be described in this formalism.

2 Preliminaries and Definitions

We assume that the reader is familiar with the basics of formal language theory and membrane computing; for more information, we refer to the monograph [10], and the handbooks [9] and [8].

In the following, we briefly review the notions and the notation we will use. An alphabet is a finite non-empty set of symbols. Given an alphabet V, we denote the set of strings over V by V^* . If the empty string, ε, is not included, then we use the notation V^+.

A finite multiset over an alphabet V is a mapping $M : V \to \mathbb{N}$ where $\mathbb{N}$ denotes the set of non-negative integers, and $M(a)$ for $a \in V$ is said to be the multiplicity of a in M. The support of M is the set $supp(M) = \{a \in V \mid M(a) \geq 1\}$. If $supp(M)$ is a finite set, then M is called a finite multiset. The set of all finite

multisets over the set V is denoted by $\mathcal{M}(V)$. We say that $a \in M$ if $a \in supp(M)$, $M_1 \subseteq M_2$ if $supp(M_1) \subseteq supp(M_2)$ and for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of two multisets over V is defined as $(M_1 \cup M_2)$ where for all $a \in V$, $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 \setminus M_2)$ where $(M_1 \setminus M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$.

In what follows, we usually enumerate the not necessarily distinct elements $a_1, \ldots, a_n$ of a multiset as $M = (a_1, \ldots, a_n)$, but the multiset M can also be represented by any permutation of a string $w = a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)} \in V^*$, where if $M(x) \neq 0$, then there exists j, $1 \leq j \leq n$, such that $x = a_j$. The empty multiset is denoted by $\emptyset$ as in the case of the empty set.

Both P systems and chemical calculi are implemented as rewriting systems. A rewriting system is a pair $\mathcal{A} = \{\Sigma, (\rightarrow_i)_{i \in I}\}$, where Σ is a set and $(\rightarrow_i)_{i \in I}$ is a set of binary relations defined on Σ. The relations $(\rightarrow_i)_{i \in I}$ are called reduction relations. In the rewriting systems considered in this paper the set § is a set of multisets of elements over an alphabet O or, in the case of the chemical formalism, multisets of terms of some calculus, and $\rightarrow$ is a single binary relation rendering multisets to multisets. It is supposed that the reduction relation $\rightarrow$ is compatible with the term formation rules in the γ-calculus, and, in the case of membrane systems, it extends itself to a relation on configurations.

2.1 The Gamma-formalism

Several attempts have been made to establish a programming language suitable for programming massively parallel architectures. The aim of the Gamma-formalism is to provide the programmer with a high-level programming language deprived of all artificial constraint for sequentiality, in which the parallel aspect is left implicit. The Gamma-formalism, as a programming language, is a tool for multiset manipulation. The only data structure, in the untyped version, is that of multisets, and the programs are collections of pairs consisting of reaction conditions and actions. The following presentation is based mainly on [2].

The meaning of the Γ-function can be defined as follows:

$$\Gamma(R, A)(M) = \begin{cases} \Gamma(R, A)((M \setminus (x_1, \ldots, x_n)) \cup A(x_1, \ldots, x_n)), \\ \qquad \text{if } x_1, \ldots, x_n \in M \text{ and } R(x_1, \ldots, x_n), \\ M \quad \text{otherwise.} \end{cases}$$

For example, the Γ-program below computes the maximal element of a set of integers M:

$$\begin{aligned} maxset(M) = \Gamma((R, A))(M) \text{ where} \\ R(x, y) = x \leq y \\ A(x, y) = (y) \end{aligned}$$

The Boolean function R describes a relation to be satisfied by the selected elements x and y. If $R(x, y)$ is true, then x and y are replaced by the result of the reaction defined by the function A. In this case, the element of the multiset

(x, y) is chosen which is not less than the other. The function R is called the reaction condition and the action A is the result of the reaction. The computation terminates when no more reactions are possible. In the present situation this leaves the computation with a one element multiset as the result, which is the maximal element of M.

The next introductory example finds the prime numbers up to a given number n:

$$\begin{aligned} \mathit{sieve}(n) = \Gamma((R, A))(2, \ldots, n) \text{ where} \\ R(x, y) = x \mathit{\ divides\ } y \\ A(x, y) = (x) \end{aligned}$$

The reaction condition is fulfilled for x and y, if y is a multiple of x. In this case the multiset (x, y) is removed from the original multiset and is replaced by the multiset (x). In effect, this means that a stable condition is reached if there are no more pairs (x, y) such that x divides y, which is satisfied if and only if the resulting multiset consists of the prime numbers not greater than n.

The definition of Γ reflects the way the program reaches its halting point: if there is at least one multiset $(x_1, \ldots, x_n)$ such that $R(x_1, \ldots, x_n)$ holds, then $\Gamma(R, A)(M)$ is applicable and the result is the same as the value determined by application $\Gamma(R, A)((M \backslash (x_1, \ldots, x_n)) \cup A(x_1, \ldots, x_n))$, if it exists.

Remark 1. Of course, there can be Γ-programs which do not terminate. In simple cases like the above ones, termination can be proven by an application of the Dershowitz–Manna lemma, see [6]. This lemma asserts that if $(S, <)$ is a well-founded ordered set with the strict (that is, irreflexive and transitive) partial-order $<$, then $\mathcal{M}(S)$, the set of all finite multisets over S with the strict partial ordering $\ll$ is also well-founded, where $\ll$ is defined as follows. Let $M, N, \in \mathcal{M}(S)$. We say that $M \ll N$ if there are X, $Y \in \mathcal{M}(S)$ such that

- $X \neq \emptyset$,
- $N = (M \backslash X) \cup Y$,
- $(\forall y \in Y)(\exists x \in X)(y < x)$.

As the well foundedness of a set means that there exists no infinite decreasing sequence of its elements, the lemma applies to the examples above, because both by the *maxset* and the *sieve* programs, the number of elements of the multisets obtained as the intermediate results of the transformation forms a strictly decreasing sequence.

2.2 Membrane Systems

A P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique and usually labeled with 1, is called the skin membrane. The membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent. If $x \in \{[_i,]_i \mid 1 \leq i \leq n\}^*$ is such a

string of matching parentheses of length $2n$, denoting a structure where membrane i contains membrane j, then $x = x_1\ [_i\ x_2\ [_j\ x_3\ ]_j\ x_4\ ]_i\ x_5$ for some $x_k \in \{[_l,]_l \mid 1 \leq l \leq n,\ l \neq i, j\}^*,\ 1 \leq k \leq 5$. If membrane i contains membrane j, and there is no other membrane, k, such that k contains j and i contains k (x_2 and x_4 above are strings of matching parentheses themselves), then we say that membrane i is the parent membrane of j. A membrane m is called elementary if it contains no membrane, in this case $x = x_1\ [_m\ ]_m\ x_2$.

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to another one. The rules are multiset rewriting rules given in the form of $u \to v$ where u, v are multisets, and they are applied in the maximal parallel manner, that is, as many rules are applied in each region as possible. The end of the computation is defined by halting: A P system halts when no more rules can be applied in any of the regions, the result is a number, the number of objects in an elementary membrane labeled as output.

Definition 1 *A P system of degree $n \geq 1$ is a construct*

$$\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, out)$$

where

- *O is an alphabet of objects,*
- *μ is a membrane structure of n membranes,*
- *$w_i \in \mathcal{M}(O),\ 1 \leq i \leq n$, are the initial contents of the n regions,*
- *$R_i,\ 1 \leq i \leq n$, are the sets of evolution rules associated to the regions, they are of the form $u \to v$ where $u \in \mathcal{M}(O)$ and $v \in \mathcal{M}(O \times TAR)$ where $TAR = \{here, out\} \cup \{in_j \mid 1 \leq j \leq n\}$, and*
- *$out \in \{1, \ldots, n\}$ is the label of an elementary membrane, the output membrane.*

The evolution rules of the system are applied in the nondeterministic, maximally parallel manner to the n-tuple of multisets of objects constituting the configuration of the system. The n-tuple $(w_1, \ldots, w_n)$ is the initial configuration of Π. For two configurations $C_1 = (u_1, \ldots, u_n)$ and $C_2 = (v_1, \ldots, v_n)$, we can obtain C_2 from C_1 by applying the rules of $R_1, \ldots, R_n$ in the following way. The application of $u \to v \in R_i$ in the region i means to remove the objects of u from u_i and add the new objects specified by v to the system. The objects of v should be added to the regions as specified by the target indicators associated to them: If v contains a pair $(a, here) \in O \times TAR$, then a is placed in region i, the region where the rule is applied. If v contains $(a, out) \in O \times TAR$, then a is added to the contents of the parent region of region i; if v contains $(a, in_j) \in O \times TAR$ for some region j which is contained inside the region i (so region i is the parent region of region j), then a is added the contents of region j.

The objects evolve simultaneously, and the rules by which they evolve are chosen nondeterministically, but in a maximally parallel manner. This means that

the rules are chosen in such a way, that in each region objects are assigned to rules, as many objects to as many rules as possible, and the assignment is maximal in the sense that no other rule can be applied to the remaining (unassigned) objects which appear unchanged in the next configuration.

A sequence of transitions between configurations is called a computation. A computation is successful if it halts, that is, if it reaches a configuration where no application of any of the rules are possible. In this case, the result is the number of objects which are present in the output region in the halting configuration.

3 Writing Gamma Programs for Membrane Systems

The Γ-formalism has been the source of inspiration for the higher-order chemical model of Banâtre et al., the so-called γ-calculus. The following presentation is based on [1].

Two syntactical elements, molecules and patterns, are defined simultaneously in the following definition.

Definition 1. *The syntactical elements of molecules, denoted by M, and patterns, denoted by P, are defined as*

$$M = x \mid \gamma(P)[M_1].M_2 \mid (M_1, M_2) \mid \langle M \rangle$$
$$P = x \mid (P_1, P_2) \mid \langle P \rangle,$$

where x is a variable standing for any molecule, $\gamma(P)[M_1].M_2$ is a γ-abstraction with reaction condition M_1 and result M_2. The operator "," is commutative and associative, and $\langle M \rangle$ is called a solution.

A solution $\langle M \rangle$ encapsulates the molecule M: molecules inside the solution are insulated from molecules outside the solution. However, the contents of solutions can be changed by reactions which occur inside the solution.

The fundamental rule of the γ-calculus is that of a reaction. Prior to defining reactions, we need some auxiliary notions.

Definition 2. *A substitution is a mapping $\phi : Var \to \mathcal{M}$, if $\mathcal{M}$ is the set of all molecules and Var represents the set of variables. We can define the application of a substitution to a molecule as follows:*

$$\phi x = \phi(x)$$
$$\phi(M_1, M_2) = \phi M_1, \phi M_2$$
$$\phi\langle M \rangle = \langle \phi M \rangle$$
$$\phi(\gamma(P)[C].M) = \gamma(P)[C].\phi' M,$$

where ϕ' is obtained from ϕ by removing from the domain all the variables which occur in P.

A molecule is termed inert if no reaction can take place within it. The first argument of *match* is a pattern, the other one is a molecule. Its value is a substitution.

Definition 3. *Let x denote a variable, P a pattern, and M a molecule. Then we define*

$$\begin{aligned} match(x, M) &= \{x \mapsto M\} \\ match(\langle P\rangle, \langle M\rangle) &= match(P, M) \text{ provided } inert(M) \\ match((P_1, P_2), (M_1, M_2)) &= match(P_1, M_1) \circ match(P_2, M_2) \\ match(P, M) &= \boldsymbol{fail} \text{ in every other case.} \end{aligned}$$

The operation $\circ$ is the function composition with the additional stipulation that $\boldsymbol{fail} \circ x = x \circ \boldsymbol{fail} = \boldsymbol{fail}$.

We are in a position now to define the main rule of the calculus, the reaction rule.

Definition 4. *Let*

$$\gamma(P)[C].M, N \to \phi M, \qquad (\gamma)$$

where $match(P, N) = \phi$ and $\phi(C) \to^* true$.

Besides the γ-rule some additional rules are applied.

Definition 5. *Let M, M_1, and M_2 be any molecule. Then we have the following rules:*

$$\frac{M_1 \to M_2}{M, M_1 \to M, M_2} \quad (locality), \quad \frac{M_1 \to M_2}{\langle M_1\rangle \to \langle M_2\rangle} \quad (solution).$$

Moreover, we identify molecules with respect to commutativity and associativity, that is, any reduction is understood modulo $\equiv$, where

$$M_1, (M_2, M_3) \equiv (M_1, M_2), M_3 \quad \text{and} \quad M_1, M_2 \equiv M_2, M_1.$$

The γ-rule is a so-called one-shot rule: after taking part in a reduction, the γ-abstraction disappears. To obtain a syntactic tool resembling the Γ-operator of the previous section, we introduce a new operator which does not vanish in the course of the reduction.

Let $replace\ P\ by\ M\ if\ C$, or $replace(P, C, M)$ in short, be an operator obeying the following reduction rule:

$$replace\ P\ by\ M\ if\ C, N \to replace\ P\ by\ M\ if\ C, \phi(M), \quad (replace)$$

where $C \to^* true$ and $match(P, N) = \phi$.

With the notation introduced, we can construct the term finding the maximal element of a given set of numbers:

Example 1. Let $M = \{3, 5, 8, 10, 12\}$. Then

$$(replace\ \langle\langle x\rangle, \langle y\rangle\rangle \text{ by } \langle y\rangle \text{ if } x \leq y, \langle 3\rangle, \langle 5\rangle, \langle 8\rangle, \langle 10\rangle, \langle 12\rangle)$$

finds the maximum element of the set M.

Observe that in the example above the integers are represented as solutions. Otherwise the term "$replace\ x, y$ by y if $x \leq y$" would match any molecules x and y. In this case, it does not seem to be a problem since $x \leq y$ makes sense if x and y are integers. As another example we compute the largest prime number up to an integer n.

Example 2.

$$\begin{aligned} largestprime(n) = \\ \text{let } sieve &= replace\ \langle\langle x\rangle, \langle y\rangle\rangle \text{ by } \langle x\rangle \text{ if } x \text{ div } y \text{ in} \\ \text{let } max &= replace\ \langle\langle x\rangle, \langle y\rangle\rangle \text{ by } \langle x\rangle \text{ if } x \leq y \text{ in} \\ &(\langle\langle 2\rangle, \langle 3\rangle, \ldots, \langle n\rangle, sieve\rangle, \gamma\langle x\rangle(x, max)) \end{aligned}$$

Observe that in this example above the pattern standing in the last γ-term is a solution $\langle x\rangle$. By the definition of *match*, only inert solutions are able to match with this pattern. This amounts to a means of governing the sequence of reductions in a molecule: first the prime numbers are selected with the term *sieve*, then the term *max* chooses the maximum among them.

Now we can turn to the problem of establishing a relation between the computations in the γ-calculus and computations in membrane systems. We consider the systems defined in Section 2.2, that is P systems of the form

$$(O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_o)$$

where n is the number of membranes in the structure. We may assume $i_o = n$, which means the output membrane is the n-th membrane.

The membrane structure can be uniquely described by setting the parent member to each element. To this end, let $par : \{1, \ldots, n\} \to \{1, \ldots, n\} \cup \{nil\}$ be the function with the interpretation: if $par(j) = k$, then the k-th membrane is the parent of the membrane numbered j. The parent of the outer membrane is nil, that is, $par(1) = nil$.

We assume, as in Definition 1, that the rules R_i belonging to membrane i are of the form $u \to v$, where $u \in \mathcal{M}(O)$ and $v \in \mathcal{M}(O \times \{here\}) \cup \mathcal{M}(O \times \{out\}) \cup \mathcal{M}(O \times \{in_j\})$, where $par(j) = i$.

First, we label the elements of each membrane, displaying explicitly the number of the membrane the element belongs to. Thus, let $lab : O \to \{(k : a) \mid 1 \leq k \leq n, a \in O\}$ be such that

$$lab(a) = (k : a),$$

if a is contained by the regions enclosed by membrane k. Let us determine how these labels are inherited through a rule $u \to v$.

Definition 6. *Let* $a \in O \times \{here\} \cup O \times \{out\} \cup O \times \{in_j\}$, *assume* $tar \in \{here, out, in_j\}$, *where* $1 \leq k \leq n$ *and* $par(j) = k$. *Then*

$$lab(k, a, tar) = \begin{cases} k & \text{if } tar = here \\ j & \text{if } tar = in_j \\ par(k) & \text{if } tar = out \text{ and } k > 1 \\ undefined & \text{if } tar = out \text{ and } k = 1. \end{cases}$$

Let k be a membrane label for some $1 \leq k \leq n$. We define a transformation $h_k(u \to v)$ of a rule $u \to v$ in region k into some γ-term. First, settle h_k for the elements in u and v. Let O be the alphabet of the membrane system. We define the image of an element of O in the corresponding γ-calculus. If $a \in O$, then

$$h_k(a) = (k : a),$$

otherwise

$$h_k(a, tar) = \begin{cases} \underline{(i : a)} & \text{where } i = lab(k, a, tar) \text{ and } tar \neq out \text{ or } k > 1, \\ \lambda & \text{if } tar = out \text{ and } k = 1. \end{cases}$$

The function h_k propagates itself to multisets, thus

$$h_k(w_1 \dots w_l) = h_k(w_1) \dots h_k(w_l),$$

if $w = w_1 \dots w_l$ and $w \in \mathcal{M}(O)$ or $w \in \mathcal{M}((O \times \{here\}) \cup \mathcal{M}(O \times \{out\}) \cup \mathcal{M}(O \times \{in_j\})$. For technical reasons we add a new constant, say, 1 to each multiset obtained from u and v, thus

$$h_k(u \to v) = replace\ (h_k(u), 1)\ with\ (h_k(v), 1).$$

Accomplishing this for every membrane labeled by k, we assign the multiset $((k : u_1), \dots, (k : u_{i_k}), (k : 1), h_k(r_1), \dots, h_k(r_{j_k}))$ to region k, where $u_1, \dots, u_{i_k}$ were the elements and $r_1, \dots, r_{j_k}$ were the rules corresponding to the region.

Let us denote by $gamma(\Pi)$ the gamma formalism assigned to the membrane system Π. The transformation above is injective, thus, given a γ-structure Γ obtained in this way, we can uniquely identify the membrane structure Π such that $gamma(\Pi) = \Gamma$. In the sequel, if no confusion occurs, we write a_k instead of $(k : a)$ and c instead of $(k : c)$, if c is a fixed element of the base set.

Example 3. Let Π be the membrane system $\Pi = (\{a, b, c\}, \mu, aa, \emptyset, R_1, \emptyset, 2)$ where $\mu = [\ [\ ]_2\ ]_1$, and $R_1 = \{a \to (a, here)(b, in_2)(c, in_2)^2,\ a^2 \to (a, out)^2\}$.

Now

$$h_1(a) = a_1,$$
$$h_1(a \to (a, here)(b, in_2)(c, in_2)^2) = replace\ (a_1, 1)\ with\ (\underline{a_1}, \underline{b_2}, \underline{c_2}^2, 1),$$
$$h_1(aa \to (a, out)^2) = replace\ (a_1, a_1, 1)\ with\ 1.$$

In what follows we simulate maximal parallel reduction sequences of the membrane system with reduction sequences in the chemical calculus γ. Besides the translations of the multiset reduction steps the process must be prepared to simulate the next maximal parallel sequence of reductions, as well. To this end, the underlined symbols must be transformed back into usual ones. Some care is needed though in the process, the replace operators assigned to rules of the membrane system should not interfere with this stage. The introduction of the additional constants 1 and 0 serves exactly this synchronization purpose. When 1 is present in the corresponding molecule, then the operators corresponding to the rules of the P system are active, and, in the case when 1 is exchanged for 0, the operators restoring the original elements of the alphabet O come into effect.

Let us define the terms giving us back the elements of the original alphabet O after simulating a maximal parallel computation sequence.

Definition 7. *Let O_{lab} be the set of elements obtained from O by completing the labeling process for every $1 \leq k \leq n$. Then*

$$d(O_{lab}) = \bigcup \{replace\ \underline{a}, 0\ with\ a, 0 \mid a \in O_{lab}\}.$$

Now we determine the term controlling the whole process.

Theorem 2. *Let $\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_n)$ be a membrane system, assume $gamma(\Pi)$ is the multiset defined as above, and $d(O_{lab})$ is the set of operators presented in the previous definition. Then, for the term*

$$M = (\langle gamma(\Pi)\rangle, replace\ \langle x, 1\rangle\ with\ \langle x, 0, d(O_{lab})\rangle,$$
$$replace\ \langle x, 0, d(O_{lab})\rangle\ with\ \langle x, 1\rangle),$$

Π comes to a halt with result $(a_1^{i_1}, a_2^{i_2}, \ldots, a_j^{i_j})$ if and only if M normalizes with $(n : a_1)^{i_1}, (n : a_2)^{i_2}, \ldots, (n : a_j)^{i_j}$ as its elements with label n.

Proof. First of all, observe that the mapping *gamma* is a bijection between membrane systems having the structure $(O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_n)$ and chemical structures over the alphabet $\bigcup_{i=1}^{n} \{h_i(O) \cup h_i(O \times \{tar\})\}$, where $tar \in \{here, out, in_j\}$ if $par(j) = i$. A little more is true. Namely, let $M \to M_1 \to \ldots$ be a reduction sequence starting from M. We call a reduction step $M_i \to M_j$ a 1-step, if the maximal solution in M_i contains 1, otherwise it is a 0-step. Let $\langle S_0\rangle$, $\langle S_1\rangle, \ldots$ be the sequence of the maximal solutions of $M \to M_1 \to M_2 \to \ldots$ and assume, for Π_0, $\Pi_1, \ldots$, $\Pi_k, \ldots$, $gamma(\Pi_0) = S_0$, $gamma(\Pi_1) = S_1, \ldots$, $gamma(\Pi_k) = S_k, \ldots$. The notion of 1-step and 0-step transforms to S_0, $S_1, \ldots$ analogously. We denote by $S' \to_1 S''$ ($S' \to_0 S''$) if a 1-step reduction from S' yields S''. Then we have the following statement.

$$S_i \to^1 S_j \text{ if and only if } \Pi_i \to \Pi_j. \tag{1}$$

The proof can be done by induction on the lengths of the reduction sequences on the two sides. Assume now $\Pi_0 \to \Pi_1 \to \Pi_2 \to \ldots \to \Pi_n$ is a reduction sequence

yielding a maximal parallel reduction step from Π_0. Then, by (1), there can be no 1-reductions starting from $gamma(\Pi_n) = S_n$. But this means S_n has arrived to a normal form. Below, for a multiset S, let $\overline{S}$ denote the multiset $S - (0,1)$. Moreover, if S is a multiset possibly with underlined elements, then let $plain(S)$ denote the multiset consisting of the same elements with the underlining removed. Thus

$$\begin{aligned}
M = {} & (\langle S_0\rangle, replace\ \langle x,1\rangle\ with\ \langle x,0,d(O_{lab})\rangle,\\
& replace\ \langle x,0,d(O_{lab})\rangle\ with\ \langle x,1\rangle) \rightarrow\\
& \ldots\\
\rightarrow {} & (\langle S_n\rangle, replace\ \langle x,1\rangle\ with\ \langle x,0,d(O_{lab})\rangle,\\
& replace\ \langle x,0,d(O_{lab})\rangle\ with\ \langle x,1\rangle) \rightarrow\\
\rightarrow {} & (\langle \overline{S}_n,0,d(O_{lab})\rangle, replace\ \langle x,1\rangle\ with\ \langle x,0,d(O_{lab})\rangle,\\
& replace\ \langle x,0,d(O_{lab})\rangle\ with\ \langle x,1\rangle) \rightarrow\\
& \ldots\\
\rightarrow {} & (\langle plain(\overline{S}_n),0,d(O_{lab})\rangle, replace\ \langle x,1\rangle\ with\ \langle x,0,d(O_{lab})\rangle,\\
& replace\ \langle x,0,d(O_{lab})\rangle\ with\ \langle x,1\rangle) \rightarrow\\
\rightarrow {} & (\langle plain(\overline{S}_n),1)\rangle, replace\ \langle x,1\rangle\ with\ \langle x,0,d(O_{lab})\rangle,\\
& replace\ \langle x,0,d(O_{lab})\rangle\ with\ \langle x,1\rangle)
\end{aligned}$$

Here, $\langle \overline{S}_n,0,d(O_{lab})\rangle$ has no choice but to reduce to $\langle plain(\overline{S}_n),0,d(O_{lab})\rangle$, which cannot be reduced any further. Then the action of $\langle plain(\overline{S}_n),0,d(O_{lab})\rangle$ and $replace\ \langle x,0,d(O_{lab})\rangle\ with\ \langle x,1\rangle$ yields $\langle plain(\overline{S}_n),1)\rangle$, which means a new maximal parallel reduction step of the membrane system obtained so far can be simulated again. This gives an account of the correspondence set up between membrane system reductions and reductions in chemical structures. □

4 Conclusion

The chemical computational paradigm describes computations as reactions between molecules which freely interact in a symbolic chemical solution. We have given a short overview of a chemical calculus from [1], and shown how computations of membrane systems can be described in this setting. Thus, we have taken some initial steps in the direction of being able to describe membrane systems using the chemical computing formalisms and being able to use the tools and techniques developed for chemical calculi to reason about membrane systems and their computations.

References

1. J.P. Banâtre, P. Fradet, Y. Radenac, Principles of chemical computing. *Electronic Notes in Theoretical Computer Science* 124 (2005) 133–147.

2. J.P. Banâtre, P. Fradet, Y. Radenac, Generalized multisets for chemical programming. *Mathematical Structures in Computer Science* 16(4) (2006), 557 – 580
3. J.P. Banâtre, D. Le Métayer, A new computational model and its discipline of programming. *Technical Report RR0566*, INRIA (1986).
4. J.P. Banâtre, D. Le Métayer, Programming by multiset transformation. *Communications of the ACM* 36 (1993), 98–111.
5. G. Berry, G. Boudol, The chemical abstract machine. *Theoretical Computer Science* 96 (1992), 217–248.
6. N. Dershowitz, Z. Manna, Proving termination with multiset orderings. *Communications of the ACM* 22(8) (1979), 465–476.
7. Gh. Păun, Computing with membranes. *Journal of Computer and System Sciences* 61 (2000), 108–143.
8. Păun, G., Rozenberg, G., Salomaa, A. (eds): The Oxford Handbook of Membrane Computing, Oxford University Press (2010)
9. Rozenberg, G. Salomaa, A. (eds): Handbook of Formal Languages, Springer Berlin (1997)
10. Salomaa, A.: Formal Languages, Academic Press, New York (1973)
11. M. Fésüs, Gy. Vaszil, Chemical programming and membrane systems. In: *Proc. 14th International Conference on Membrane Computing*, Institute of Mathematics and Computer Science, Academy of Moldova, 2013, 313–316.

The Reduction Problem in CUDA and Its Simulation with P Systems

Rodica Ceterchi[1], Miguel Ángel Martínez-del-Amor[2], Mario J. Pérez–Jiménez[2]

[1] Faculty of Mathematics and Computer Science
University of Bucharest
14 Academiei st. 010014 Bucharest, Romania
E-mail: rc@fmi.unibuc.ro, rceterchi@gmail.com
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es, marper@us.es

Summary. We introduce P systems with dynamic communication graphs which simulate the functioning of the CUDA architecture when solving the parallel reduction problem.

1 Introduction

Introduced in [13], P systems are powerful computational devices, with a high degree of parallelism, whose functioning is inspired by biological processes at the level of the cells, and of their membranes ([13],[14]). Among these processes, rewriting and communication play an important role.

It is of interest to compare P systems with other, classical, parallel paradigms. We have begun such a study in the form of simulating parallel classical architectures with P systems, in particular, the perfect shuffle architecture [4], [5] and the mesh architecture [6]. The reduction problem, being one of the most simple, primary ones to be solved in different contexts, was used as an illustration.

In [7] some general guidelines were developed along which a wide class of parallel architectures can be simulated with P systems. P systems with dynamic communication graphs were introduced. In their functioning, rewriting steps and communication steps are separated and made more visible. They differ from the systems introduced in [3] in that the communication graphs are inspired by the particular parallel network architecture being simulated.

CUDA stands for *Compute Unified Device Architecture* [15, 10], and is a technology proprietary of NVIDIA Corp. Since its introduction in 2007, CUDA has

allowed programmers to take advantage of the inherent parallel architecture of GPUs, which ranges from 240 cores (Tesla C1060, released on 2008) to 2880 cores (Tesla K40, released on 2013). This is performed by using a threaded, shared-memory, abstracted model of the GPU, which is implemented by C/C++ extensions. CUDA has helped to establish GPU computing [9] as a sub-framework of High Performance Computing. In fact, it has been successfully applied to a broad spectrum of research areas, including Systems Biology and Population Dynamics [12], and Membrane Computing [1, 2, 11], among others.

In the present paper we propose to simulate with P systems the reduction problem as solved in CUDA. Section 2 is devoted to the presentation of several improved versions of solving the reduction problem in CUDA. Section 3 is devoted to the presentation of its simulation with P systems with dynamic communication graphs.

2 Solving the Reduction Problem in CUDA

The *GPU* (Graphics Processor Unit) is the core of graphics cards. A GPU today contains thousands of computing processors devoted for graphics. However, novel techniques enable programmers to take advantage of this highly parallel architecture for scientific computing. These are called *GPGPU* (General Purpose computing on the GPU) [9].

A new era of GPGPU started with the introduction of *CUDA* (Compute Unified Device Architecture) [15, 10] by NVIDIA. It offers a programming model that abstracts the GPU architecture to programmers, so it is enough to learn some extensions to C/C++ language (CUDA extensions), whereas the CUDA driver will execute the code on the GPU. In the following sections we will introduce some concepts and terminology of CUDA which are necessary to understand the work presented in this paper.

2.1 CUDA programming model

The CUDA programming model assumes that the CPU (or *host*) takes control of the execution flow, and permit the GPU (or *device*) to run many instances of the same code in parallel. This code is called *kernel*, and it is executed by a *grid* of *threads*. Typically, a grid is composed of thousands of threads, since the creation of a sufficient number of threads to use all hardware resources requires a large amount of data parallelism. The threads are arranged within the grid in a two-level hierarchy, as seen in Figure 1. At the higher level, each grid consists of one or more *thread blocks*. At the lower level, each block is organized as a three dimensional array of threads. All blocks in a grid have the same number and organization of threads. Each block is identified by a two dimensional identifier, and each thread within its block by a three dimensional identifier (ID). Therefore, any thread can be unequivocally identified by the union of both thread and thread

block identifiers. The execution of threads inside a block can be synchronized by *barrier* operations (`__syncthreads()`), and threads of different blocks can be synchronized only by finishing the execution of the kernel.

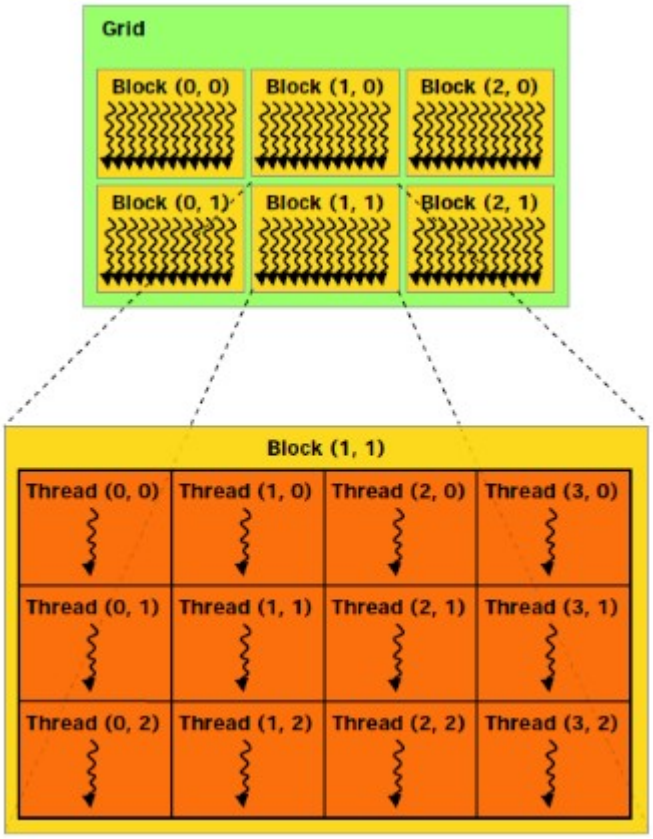


Fig. 1. Threading model in CUDA. Threads are executed in a grid, and they are organized in blocks.

The memory hierarchy is explicitly and manually managed in CUDA. This memory model is composed in several levels, each one offering different speeds and storage properties. We highlight the two most important ones: *global* memory and *shared* memory. Global memory is the largest but the slowest memory in the system. It is accessed by the host (where the input and output data are allocated) and by any thread in execution. Shared memory is the smallest but fastest memory. It is accessed by threads belonging to the same block. Normally, performance of CUDA applications depends on how much shared memory is exploited. Thus, an efficient way to structure an algorithm is as follows:

1. The threads of each block read its corresponding data portion from global memory to shared memory (which is inevitable because the host only can put the data in global memory).
2. Threads work with the data directly on the shared memory.
3. Threads copy these data back to global memory (so the host can retrieve the result).

2.2 Modern GPU architecture

The GPU architecture has evolved in the last years, offering even more compute capabilities. In general terms, it consists of a scalable processor array, organized in *Streaming Multiprocessors (SMs)* of *Streaming Processors (SPs, or cores)*. The number of them depends on the GPU. SMs are based on the *SIMT (Single-Instruction Multiple-Thread)* model. Basically, in the SIMT model all the threads execute the same instruction on different piece of data. SMs create, manage, schedule and execute threads in groups of 32 threads (which is the branching granularity of NVIDIA GPUs). This set of 32 threads is called *warp*, and each SM can handle many of them. Individual threads of the same warp must start together at the same program address. However, they are free to branch and execute independently, but at cost of serialization and performance (in fact, SIMT is really applied to the warp). If a warp is broken (because of branching or memory stall), the real parallelism in CUDA is not achieved.

2.3 Performance considerations

Although CUDA programming model is flexible enough to run any kind of algorithm, the achieved performance depends on how the programmer had designed the code, and on the target GPU running the program. A CUDA programmer has to perfectly know the CUDA programming model, but also the idea of the GPU architecture, since it provides the restrictions to be considered in order to achieve peak performance. There are several strategies to accomplish it. Next, we stand out two of them:

- *Emphasize parallelism*: the warp is the branching granularity on CUDA; that is, the parallelism unit. Thus, warps must be maximized with active threads, but minimizing branch divergence between thread: they must be executing the same instruction simultaneously to reach peak performance.
- *Exploit memory bandwidth*: the peak bandwidth of using both global and shared memories is achieved mainly by an access pattern: coalesced access to contiguous (aligned) memory positions. Data is transferred from memory to the GPU hardware in blocks, which is formed by contiguous bytes in memory. Thus, we must maximize these blocks with the access to contiguous memory addresses by contiguous threads within a warp.

2.4 Parallel Reduction in CUDA

The reduction problem consists in applying an operator to a set of elements. Let us assume a set of n elements $\{a_1, \ldots, a_n\}$, and the binary and associative, reduction operator $\oplus$. The result of applying reduction to the set of elements is another element $a = a_1 \oplus a_2 \oplus \ldots a_n$. Reduction is a well-known primitive in Parallel Computing, since it resides inside many important algorithms. For example, it can be used to compute the sum or the maximum of an array of numbers. Nowadays, reduce is part of the most used algorithm in Big Data and No-SQL data bases, which is Map-Reduce.

A common way to solve this problem in parallel is by using a tree, in which partial solutions are computed to reach the final one. The time complexity of this solution is $O(logn)$. The process is summarized in figure 2.

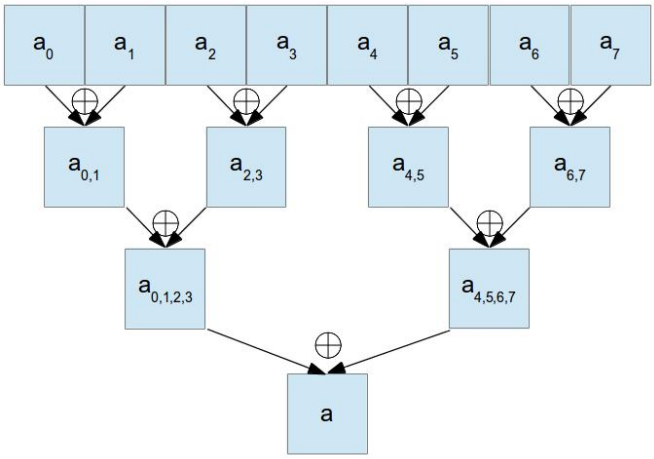


Fig. 2. Scheme of a parallel reduction solution

The most popular CUDA implementations for the reduction primitive can be found on the speech given by Harris [8] in 2007. This document introduces seven kernels from a didactic perspective, in a performance-increasing order. Next, we discuss the first four ones.

Interleaved addressing

Let us assume that the input array of elements is of size n, and that the maximum amount of threads per block is nt. That means that we will need to use $nb = \frac{n}{nt}$ thread blocks to process the whole array. However, if each block compute reduce for nt elements, what would we do with the nb partial results? The answer is to have a second kernel, with a single block having nb threads, which will compute

again reduce. It is straightforward to add more kernels in this way until having $nb < nt$.

In what follows, we will assume that the size of the input array of elements is less than nt (maximum number of threads per block). This will mean that just one thread block is enough to compute reduce. We will disregard the second kernel for partial results.

A naive implementation of the tree solution in CUDA is to launch n threads and correspond each one with an element. First, each thread with even ID (called active threads) will compute the partial result with the next element, and the process will continue by halving the number of active threads. This process, called *reduce0* (interleaved addressing), is shown in Figure 3 (white-arrowed threads are inactive).

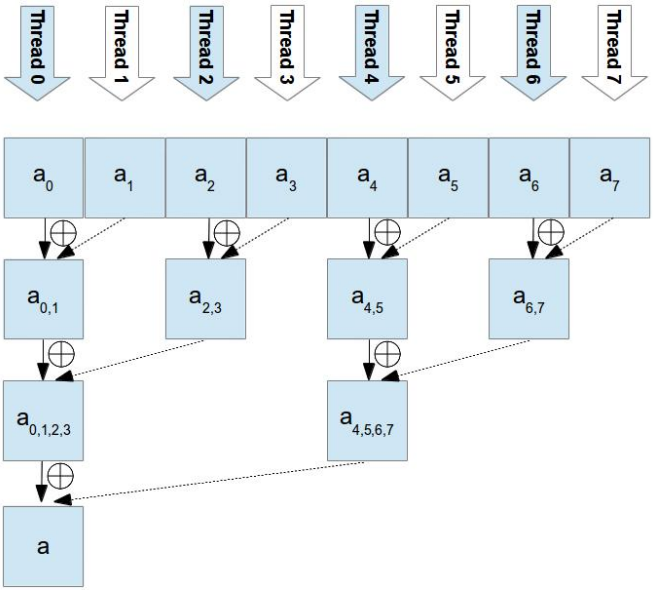


Fig. 3. Reduce0: interleaved addressing.

The main drawbacks of this solution are:

- Warps are not fulfilled. Since the addressing is interleaved, warps can be filled by active threads up to the half. Therefore, we are not maximizing memory.
- Access to memory is also interleaved, so the peak bandwidth cannot be reached.

Sequential addressing

A solution to the drawbacks found in interleaved addressing is to compact the memory accesses to contiguous threads. In order to implement this approach, it

will be necessary to change the tree of the reduce primitive solution. Now, the first half of threads will access to the first half of the array to compute the partial solutions with the second half. It can be seen that the access is coalesced in this way, as well as warps are also fulfilled. Figure 4 shows the scheme of this approach, called *reduce 3*, sequential addressing. Again, white-arrowed threads are inactive in the beginning of the process.

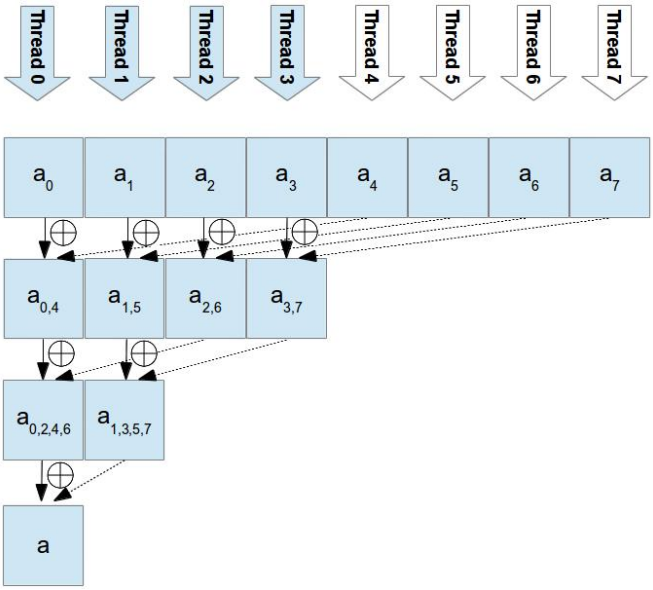


Fig. 4. Reduce3: sequential addressing.

As it can be noticed, the main drawback of this solution is that half of the threads are idle on first loop iteration, what is a waste of resources.

First add during load

The third approach, called *reduce4* first add during load, is based on taking advantage of the threads which are inactive at the beginning of reduce3. The idea is to halve the number of blocks, and to use all the threads at the beginning to apply the reduction operation to the corresponding element with the element that would have corresponded to the avoided extra block. That is, reduce4 will compute first the array $\{a_{0,8}, a_{1,9}, a_{2,10}, a_{3,11}, a_{4,12}, a_{5,13}, a_{6,14}, a_{7,15}\}$, and proceed as in reduce3.

3 The Simulation with P Systems

In this section we use the formal tools developed in [7] to produce a straightforward simulation with P systems of the reduction problem solved in CUDA as presented in section 2. In [7] only **SIMD** machines were considered, and algorithms for which communication took place only via a network of communication and not via a shared memory. The present case is different, we have communication via shared memory.

As a first step we construct for the **CUDA** model a P system $\Pi(\mathbf{C})$ in the spirit of Theorem 5 of [7]. The system must reflect in its membrane structure the particular CUDA architecture. As a second step, we follow Theorem 7 of [7], and construct $\Pi(\mathbf{C}, Y)$ for Y a reduction algorithm. We must specify for each algorithm the specific sequence of pairs $(graph, rules)$ which compose $R_\mu(Y)$.

Let $Graphs$ denote the set of all possible graphs having n vertices labeled $P_1, \cdots, P_n$. Having fixed the vertices, each element of $Graphs$ will be uniquely identified by the specific set of edges.

A distinguished element of $Graphs$ is the *identity* graph, denoted in the sequel Id: (the set of vertices is fixed as mentioned above) the set of edges is defined as

$$Id = \{(i, i) \mid 1 \leq i \leq n\}.$$

Another distinguished element of $Graphs$ is the *total graph*, denoted G_{total}: the set of edges is defined as

$$G_{total} = \{(i, j) \mid 1 \leq i, j \leq n\}.$$

One can also consider the *strict total graph*, denoted G^+_{total}: with set of edges defined as

$$G^+_{total} = \{(i, j) \mid 1 \leq i, j \leq n, i \neq j\} = G_{total} \setminus Id.$$

We recall from [7] the following two definitions.

Definition 3.1 *A P system* with dynamic communication graphs *is a construct*

$$\Pi =< V, P_1, \cdots, P_n, R_\mu >,$$

where $P_1, \cdots, P_n$ are elementary membranes, and V is an alphabet of symbols used to codify the contents of the membranes.

R_μ is a set of pairs $[graph, rules]$, with $graph \in Graphs$ and such that:

(i) if $graph \subseteq Id$ then its associated rules are rewriting rules;
(ii) if $graph \subseteq G^+_{total}$ then its associated rules are communication rules.

Definition 3.2 *A P system with dynamic communication graphs will be called* with finite sequential support *iff the set R_μ is both finite and totally ordered, i.e., if it is a finite sequence.*

Let V be an alphabet of symbols with which we will codify the contents of the membranes. An integer n will be codified as a^n (n apparitions of the symbol a, with $a \in V$). We assume we solve the reduction problem for a binary commutative and associative operation $*$, and we assume we can compute $n * m$ inside a membrane by rewriting: i.e. we have symbols $a, b \in V$ and a rewriting rule $r_*(a, b)$ such that $r_*(a, b)(a^n b^m) = a^{n*m}$.

We associate a hierarchical membrane structure to the CUDA components in the following manner: (1) each thread is an elementary membrane; (2) each block is a membrane containing the elementary ones associated to its threads; (3) the global memory is a separate elementary membrane. The presentation of this membrane structure is

$$\mu = (B_0(P_{00}, P_{01}, \cdots P_{0n}), \cdots, B_k(P_{k0}, P_{k1}, \cdots P_{kn}), M_g),$$

where $n = 2^t$ is the number of threads per block, M_g is the global memory membrane, P_{ij} is the membrane corresponding to thread j of block i, and when we reason inside a block the subscript corresponding to the block may be omitted.

If we have a set R_μ of pairs $(graph, rules)$ to obey the conditions of the definition, then the construct

$$\begin{aligned} \Pi(\mathbf{C}) &= (V, (B_0(P_{00}, P_{01}, \cdots P_{0n}), \cdots, B_k(P_{k0}, P_{k1}, \cdots P_{kn}), M_g), R_\mu) \\ &= (V, \mu, R_\mu) \end{aligned}$$

is a P system with dynamic communication graphs.

Here a discussion may start, comparing the P systems devised in [7] for **SIMD-X** machines, and a potential similar candidate for the CUDA paradigm. Such a candidate depends on a good definition for R_μ, or, at least, the formulation of criteria for 'admissible' candidates.

We open the way for this discussion, which is also a reflection on the power and the limitations of the formalism introduced in [7], by simulating the solving of the reduction problem in CUDA. More precisely, in the following we construct sets $R_\mu(Y)$ with the property that $\Pi(\mathbf{C}, Y) = (V, \mu, R_\mu(Y))$ is a P system with dynamic communications graph, with finite sequential support, which simulates Y, a reduction algorithm among the ones presented in Section 2.

The admissible communication graphs will have to reflect the communication properties of CUDA. We will have communication edges between M_g and each thread membrane P_{ji} to simulate the reading from and the writing to global memory. Between the individual elementary membranes which simulate the threads we can have communication only inside the same block, so the communication graph will have separate connected components for each block. Rewriting rules inside membranes will be associated to subgraphs of Id and communication rules to subgraphs of G^+_{total}.

We use the shorthand notation (A, B, x) for symbol x traveling on an oriented edge (A, B) from A to B, (G, x) for symbol x travelling on all oriented edges of G,

and $(\{G_j\}_j, x)$ for symbol x traveling on all oriented edges of the family of graphs $\{G_j\}_j$.

The membrane M_g contains integers n_{ij} each codified with a symbol a_{ij}

$$M_g = \{a_{ij}^{n_{ij}} \mid i = 0, \cdots, k, j = 0, \cdots, n\}$$

The graph for loading from global memory will be $\{(M_g, P_{ij}) \mid i, j\}$, and writing from thread P_{ij} to global memory will use the edge (P_{ij}, M_g).

Loading the integers from global memory into the membranes corresponding to threads will be simulated by the sequence

$$(\{(M_g, P_{ij}, a_{ij})\}_{ij}, \{(P_{ij}, a_{ij} \to a)\}_{ij}),$$

where the first step is a communication step, and the second a rewriting step.

Writing to global memory from block k will be simulated by the sequence

$$((P_{k0}, a \to a_{k0}), (P_{k0}, M_g, a_{k0}))$$

where the first step is a rewriting step, and the second a communication step.

We now construct the graph for *interleaved addressing* and the sequence of rules which simulate the procedure reduce0 of section 2. We assume we are inside a block and we omit the block index. For a fixed stride s the graph of interleaved addressing will be

$$G_s = \{(P_{i+s}, P_i) \mid i \bmod (2s) = 0\}.$$

On one edge of this graph the sequence of rules to be applied is

$$((P_{i+s}, a \to ab), (P_{i+s}, P_i, b), (P_i, r_*(a, b))).$$

We first rewrite a to ab in P_{i+s}, then the b symbol travels to P_i, and finally in P_i the application of the rewriting rule r_* produces the desired result.

For the entire block, the sequence of rules for stride s will be

$$R_s = ((\{P_{i+s}\}_i, a \to ab), (G_s, b), (\{P_i\}_i, r_*(a, b))).$$

To finish the simulation of reduce0 we have to iterate R_s corresponding to the sequence of strides for this case, i.e. we consider the sequence

$$(R_s \mid s = 1, s <= n, s = 2 * s).$$

For *sequential addressing* the communication graph inside a block is

$$G'_s = \{(P_{i+s}, P_i) \mid i = 0, 1, \cdots, s - 1\}.$$

For the entire block, the sequence of rules for stride s will be

$$R'_s = ((\{P_{i+s}\}_i, a \to ab), (G'_s, b), (\{P_i\}_i, r_*(a, b))).$$

To finish the simulation we iterate R'_s corresponding to the sequence of strides for this case, i.e. we consider the sequence

$$(R'_s \mid s = n = 2^t, s > 0, s = s \text{ div } 2).$$

We can analogously simulate the remaining versions of the procedure reduce of section 2. We illustrate with the improvement *first add during load*.

In this case we halve the number of blocks, and thus of threads. Equivalently, we can consider that M_g contains a double number of integers, codified with a number of symbols doubled compared to the number of threads. We denote a_{ij} and b_{ij} the symbols which will correspond to the thread P_{ij}. The previous loading sequence will be replaced by the load-and-add sequence

$$(\{(M_g, P_{ij}, a_{ij}, b_{ij})\}_{ij}, \{(P_{ij}, r_*(a_{ij}, b_{ij}), a_{ij} \to a)\}_{ij}).$$

References

1. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with Active Membranes on CUDA, *Briefings in Bioinformatics*, **11**, 3 (2010), 313–322.
2. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Simulating a P system based efficient solution to SAT by using GPUs, *Journal of Logic and Algebraic Programming*, **79**, 6 (2010), 317–325.
3. R. Ceterchi, C. Martín–Vide: Dynamic P Systems. In *"Membrane Computing", International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002, Revised Papers*, (G. Păun, G. Rozenberg, A. Salomaa, C. Zandron eds.), LNCS 2597, Springer 2003, p. 146-186
4. R. Ceterchi, M.J. Pérez Jiménez: Simulating Shuffle–Exchange Networks with P Systems. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, F. Sancho and A. Romero, eds.), Report RGNC 01/04, University of Seville 2004, 117-129.
5. R. Ceterchi, M.J. Pérez Jiménez: A Perfect Shuffle Algorithm for Reduction Processes and its Simulation with P Systems. In *Proceedings of the International Conference on Computers and Communications ICCC 2004* (I. Dzitac, T. Maghiar, C. Popescu, eds.), Editura Univ. Oradea, 2004, 92-97
6. R. Ceterchi, M.J. Pérez Jiménez: On Two-Dimensional Mesh Networks and Their Simulation with P Systems. In *Membrane Computing, 5^{th} International Workshop, WMC 2004, Revised Selected and Invited Papers* (G. Mauri, Gh. Păun, M. J. Pérez–Jiménez, G. Rozenberg, A. Salomaa, eds.), LNCS 3365 (2005), 259-277.
7. R. Ceterchi, M.J. Pérez Jiménez: On Simulating a Class of Parallel Architectures, *International Journal of Foundations of Computer Science, Vol. 17, No. 1 (2006)* 91–110
8. M. Harris: Optimizing Parallel Reduction in CUDA, NVIDIA Developer Technology (2007).
9. M. Harris. Mapping computational concepts to GPUs, *ACM SIGGRAPH 2005 Courses*, NY (USA), 2005.

10. D. Kirk, W. Hwu. *Programming Massively Parallel Processors: A Hands On Approach*, MA (USA), 2010.
11. M.A. Martínez-del-Amor, J. Pérez-Carrasco, M.J. Pérez-Jiménez. Characterizing the parallel simulation of P systems on the GPU. *International Journal of Unconventional Computing*, **9**, 5-6 (2013), 405-424.
12. M.A. Martínez-del-Amor, I. Pérez-Hurtado, A. Gastalver-Rubio, A.C. Elster, M.J. Pérez-Jiménez. Population Dynamics P systems on CUDA. In *10th Conference on Computational Methods in Systems Biology, CMSB2012*, (D. Gilbert, M. Heiner, eds.), LNBI 7605 (2012), 247-266.
13. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998
14. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
15. *NVIDIA CUDA website*, 2014. `https://developer.nvidia.com/cuda-zone`

Towards P Colonies Processing Strings

Luděk Cienciala[1], Lucie Ciencialová[1], Erzsébet Csuhaj-Varjú[2]

[1] Institute of Computer Science
and
Research Institute of the IT4Innovations Centre of Excellence,
Silesian University in Opava, Czech Republic
{ludek.cienciala, lucie.ciencialova,}@fpf.slu.cz
[2] Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
csuhaj@inf.elte.hu

Summary. In this paper we introduce and study P colonies where the environment is given as a string. These variants of P colonies, called Automaton-like P systems or APCol systems, behave like automata: during functioning, the agents change their own states and process the symbols of the string. After introducing the concept of APCol systems, we examine their computational power. It is shown that the family of languages accepted by jumping finite automata is properly included in the family of languages accepted by APCol systems with one agent, and it is proved that any recursively enumerable language can be obtained as a projection of a language accepted by an Automaton-like P colony with two agents.

1 Introduction

P colonies were introduced in [13] as formal models of a computing device combining properties of membrane systems and distributed systems of formal grammars called colonies. The concept was inspired by the structure and functioning of a community of living organisms in a shared environment (for more information consult [14]).

In the basic model, the cells or agents are represented by a finite collection of objects and rules for processing these objects. The agents are restricted in their capabilities, i.e., only a limited number of objects, say, k objects, are allowed to be inside any cell during the functioning of the system. These objects represent the current state of the agents, in other terms the current contents of the cells. The rules of the cells are either of the form $a \to b$, specifying that an internal object a is transformed into an internal object b, or of the form $c \leftrightarrow d$, specifying that an internal object c is exchanged by an object d in the environment. After applying these rules in parallel, the state of the agent will consist of objects b, d. Each agent is associated with a set of programs composed of such rules.

The agents of a P colony perform a computation by synchronously applying their programs to the objects representing the state of the agents and objects in the environment. At the beginning of the computation, executed by a given P colony of capacity k, i.e., where any agent has at most k symbols inside, the environment contains arbitrarily many copies of a distinguished symbol e, called the environmental symbol (and no more symbols); furthermore, each cell contains k copies of e. When a halting configuration is reached, that is, when no more rules can be applied, the result of the computation is read as the number of certain types of objects in the environment.

P colonies have been extensively examined during the years. For example, it was shown that these systems are computationally complete computing devices even with very restricted size parameters and with other (syntactic or functioning) restrictions [1, 2, 4, 5, 6, 7, 9, 10].

According to the the basic model, the impact of the environment on the behavior of the P colony is indirect. To describe the situation when the behavior of the components of the P colony is influenced by direct impulses coming from the environment step-by-step, the model was augmented with a string put on an input tape to be processed by the P colony [3]. These strings corresponds to the impulse sequence coming from the environment. In addition to their rewriting rules and the rules for communicating with the environment, the agents have so-called tape rules which are used for reading the next symbol on the input tape. This is done by changing one of the objects of the current state of the agents to the object corresponding to the current input symbol on the tape. The symbol is said to be read if at least one agent applied its corresponding tape rule. The model, called a P colony automaton or a PCol automaton, combines properties of standard finite automata and standard P colonies. The P colony automaton starts working with a string on its input tape (the input string) and with initial multisets of objects in its cells. The input string is accepted if it is read by the system and the P colony is in an accepting configuration (in an accepting state). It was shown that P colony automata are able to describe the class of recursively enumerable languages, taking various working mode into account.

In this paper we make one step further in combining properties of P colonies and automata. While in the case of PCol automata the behaviour of the system is influenced both by the string to be processed and the environment consisting of multisets of symbols, in the case of Automaton-like P colonies or APCol systems, for short, introduced in this article, the whole environment is a string. The interaction between the agents in the P colony and the environment is realized by exchanging symbols between the objects of the agents and the environment (communication rules), and the states of the agents may change both via communication and evolution; the latter one is an application of a rewriting rule to an object. The distinguished symbol, e (in the previous models the environmental symbol) have a special role: whenever it is introduced in the string by communication, the corresponding input symbol is erased.

The computation in APCol systems starts with an input string, representing the environment, and with each agents having only symbols e in its state. Every computational step means a maximally parallel action of the active agents: an agent is active if it is able to perform at least one of its programs, and the joint action of the agents is maximally parallel if no more active agent can be added to the synchronously acting agents. The computation ends if the input string is reduced to the empty word, there are no more applicable programs in the system, and meantime at least one of the agents is in so-called final state.

After defining the model, we examined the computational power of APCol systems. We proved that the family of languages accepted by jumping finite automata is properly included in the family of languages accepted by APCol systems with one agent and it was shown that any recursively enumerable language can be obtained as a projection of a language accepted by an Automaton-like P colony with two agents. We also provided several examples to demonstrate the behaviour of an APCol system.

2 Definitions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory and membrane computing. For further details we refer to [11] and [17].

For an alphabet Σ, the set of all words over Σ (including the empty word, ε), is denoted by Σ^*. We denote the length of a word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$.

A multiset of objects M is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : V \to N$; f assigns to each object in V its multiplicity in M. The set of all multisets with the set of objects V is denoted by V°. The set V' is called the support of M and denoted by $supp(M)$. The cardinality of M, denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects $V' = \{a_1, \ldots a_n\}$ can be represented as a string w over alphabet V' with $|w|_{a_i} = f(a_i);\ 1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M, and ε represents the empty multiset.

In the following we introduce the concept of an Automaton-like P colony (an APCol system, for short) where the environment of the agents is given in the form of a string.

As in the case of standard P colonies, agents of the APCol systems contain objects, each being an element of a finite alphabet. With every agent, a set of programs is associated. There are two types of rules in the programs. The first one, called an evolution rule, is of the form $a \to b$. It means that object a inside of the agent is rewritten (evolved) to the object b. The second type of rules, called a communication rule, is in the form $c \leftrightarrow d$. When this rule is performed, the object c inside the agent and a symbol d in the string are exchanged, so, we can say that

the agent rewrites symbol d to symbol c in the input string. If $c = e$, then the agent erases d from the input string and if $d = e$, symbol c is inserted into the string.

An Automaton-like P colony works successfully, if it is able to reduce the given string to ε, i.e., to enter a configuration where at least one agent is in accepting state and the processed string is the empty word.

Definition 1. *An Automaton-like P colony (an APCol system, for short) is a construct*

$$\Pi = (O, e, A_1, \ldots, A_n), \text{ where}$$

- O *is an alphabet; its elements are called the objects,*
- $e \in O$, *called the basic object,*
- $A_i,\ 1 \leq i \leq n$, *are agents. Each agent is a triplet* $A_i = (\omega_i, P_i, F_i)$, *where*
  - ω_i *is a multiset over* O, *describing the initial state (content) of the agent,* $|\omega_i| = 2$,
  - $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ *is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:*
    - $a \to b$, *where* $a, b \in O$, *called an evolution rule,*
    - $c \leftrightarrow d$, *where* $c, d \in O$, *called a communication rule,*
  - $F_i \subseteq O^*$ *is a finite set of final states (contents) of agent* A_i.

In the following we explain the work of an Automaton-like P colony; to help the easier reading we provide only the necessary formal details.

During the work of the APCol system, the agents perform programs. Since both rules in a program can be communication rules, an agent can work with two objects in the string in one step of the computation. In the case of program $\langle a \leftrightarrow b; c \leftrightarrow d\rangle$, a substring bd of the input string is replaced by string ac. If the program is of the form $\langle c \leftrightarrow d; a \leftrightarrow b\rangle$, then a substring db of the input string is replaced by string ca. This means that the agent can act only in one place in the one step of the computation and what happens to the string depends both on the order of the rules in the program and on the interacting objects. In particular, we have the following types of programs with two communication rules:

- $\langle a \leftrightarrow b; c \leftrightarrow e\rangle$ - b in the string is replaced by ac,
- $\langle c \leftrightarrow e; a \leftrightarrow b\rangle$ - b in the string is replaced by ca,
- $\langle a \leftrightarrow e; c \leftrightarrow e\rangle$ - ac is inserted in a non-deterministically chosen place in the string,
- $\langle e \leftrightarrow b; e \leftrightarrow d\rangle$ - bd is erased from the string,
- $\langle e \leftrightarrow d; e \leftrightarrow b\rangle$ - db is erased from the string,
- $\langle e \leftrightarrow e; e \leftrightarrow d\rangle$; $\langle e \leftrightarrow e; c \leftrightarrow d\rangle$, ...- these programs can be replaced by programs of type $\langle e \to e; c \leftrightarrow d\rangle$.

At the beginning of the work of the APCol system (at the beginning of the computation), there is an input string placed in the environment, more precisely, the environment is given by a string ω of objects which are different from e.

This string represents the initial state of the environment. Consequently, an initial configuration of the Automaton-like P colony is an $(n+1)$-tuple $c = (\omega; \omega_1, \ldots, \omega_n)$ where w is the initial state of the environment and the other n components are multisets of strings of objects, given in the form of strings, the initial states the of agents.

A configuration of an Automaton-like P colony Π is given by $(w; w_1, \ldots, w_n)$, where $|w_i| = 2,\ 1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent and $w \in (O - \{e\})^*$ is the string to be processed.

At each step of the (parallel) computation every agent attempts to find one of its programs to use. If the number of applicable programs is higher than one, the agent non-deterministically chooses one of them. At one step of computation, the maximal possible number of agents have to be active, i.e., have to perform a program.

By applying programs, the Automaton-like P colony passes from one configuration to another configuration. A sequence of configurations started from the initial configuration is called a computation. A configuration is halting if the AP-Col system has no applicable program. A computation is called accepting if and only if at least one agent is in final state and the string to be processed is ε. Hence, the string w is accepted by the Automaton-like P colony Π if there exists a computation by Π such that it starts in the initial configuration $(\omega; \omega_1, \ldots, \omega_n)$ and the computation ends by halting in the configuration $(\varepsilon; w_1, \ldots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

3 Computational power of Automaton-like P colonies

The behaviour of Automaton-like P colonies is similar to the functioning of jumping finite automata. The jumping finite automaton is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where the meaning of Q, Σ, δ, q_0 and F is the same as in the case of traditional finite automaton with ε-steps (the set of states, the input alphabet, the transition function, the initial state, and the set of final states). The dissimilarity of the two computing devices is in the way of performing a computational step. The computation starts in a random cell of the input tape. After reading the input symbol in the cell and changing the state of the automaton, the reading head is allowed to jump to some random location on the tape. If the symbol is read, then it is erased from the tape. The notion of a jumping finite automaton was introduced in [15], we refer to this seminal article for the precise details.

Although non-trivial languages can be recognized by jumping finite automata, the following languages cannot be accepted by them: $L_1 = \{ab\},\ L_2 = \{a^n b^n \mid n \geq 0\},\ L_3 = \{a^n b^n c^n \mid n \geq 0\}$. But, jumping finite automata accept $L_4 = \{ab, ba\}, L_5 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\},\ L_6 = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$. It is shown that the family of languages accepted by jumping finite automata is included in the family of context-sensitive languages.

Theorem 1. *For every jumping finite automaton $M = (q, \Sigma, \delta, q_0, F)$ we can construct an Automaton-like P colony $\Pi = (O, e, A)$ such that $L(M) = L(\Pi)$ holds.*

Proof. Let $M = (q, \Sigma, \delta, q_0, F)$ be a jumping finite automaton. We construct an Automaton-like P colony Π with one agent A which simulates every computation of M and only that. The simulation is as follows:

- The current state of M is stored as object inside the agent of Π, i.e., it corresponds to a state of the agent.
- One step of computation of M (except of ε-step) is simulated by two computation steps of Π. In the first step, the agent changes the object corresponding to the state of M and replaces the symbol on the tape by object e - erases the symbol has been read from the input string. In the second step, the agent prepares itself for the simulation of the next step of computation of M - it rewrites the consumed symbol from the tape to e.

Formally, we construct the Automaton-like P colony $\Pi = (A, e, B)$ as follows:

1. $O = Q \cup \Sigma \cup \{e\}$;
2. $A = (q_0 e, P_1, \{q_f e \mid q_f \in F\})$;
3. $P_1 = \{$
   $\langle q \to q', e \leftrightarrow a \rangle$ for every $q, q' \in Q, a \in \Sigma$ such that $q' \in \delta(q, a)$
   $\langle q' \to q', a \to e \rangle$ for every $q' \in Q, a \in \Sigma$
   $\langle q \to q', e \to e \rangle$ for every $q, q' \in Q$ such that $q' \in \delta(q, \varepsilon)$
   $\}$

The Automaton-like P colony starts processing an input string w in initial configuration $(q_0 e; w)$ that corresponds to the initial configuration of jumping finite automaton M (q_0, w).

If M is in the configuration (q, uav), $q \in Q, u, v \in \Sigma^*, a \in \Sigma$ and it performs computational step reaching state q' such that $q' \in \delta(q, a)$, then it enters configuration (q', uv). If the APCol system Π is in the corresponding configuration $(qe; uav)$, then the agent has an applicable program $\langle q \to q', e \leftrightarrow a \rangle$. After executing this program, Π enters configuration $(q'a; uv)$ and in the next step the program $\langle q' \to q', a \to e \rangle$ must be performed. The configuration of Π is $(q'e, uv)$ that corresponds to configuration of M. Then the APCol system Π is prepared to simulate the next step of computation of jumping finite automaton M.

If the automaton M is in the configuration (q, u), $q \in Q, u \in \Sigma^*$ and it performs computational step reaching state q' such that $q' \in \delta(q, \varepsilon)$, then it enters configuration (q', u). If the APCol system Π is in the corresponding configuration $(qe; u)$, then the agent has applicable program $\langle q \to q', e \to e \rangle$. After execution of this program the system is in the configuration $(q'; u)$. This configuration corresponds to configuration of the jumping finite automaton M.

The automaton M accepts input string w iff it passes from the initial configuration (q_0, w) to one of final configuration (q_f, ε). This computation corresponds to computation in Π that starts in the initial configuration $(q_0 e; w)$ and ends in one of final configurations $(q_f e; \varepsilon)$. The computation of Π is halting if and only

if the computation in M is halting, too. Hence the Automaton-like P colony Π accepts string w if only if jumping finite automaton M accepts w. □

Now we show that Automaton-like P colonies are able to accept languages that jumping finite automata cannot.

Example 1. Let $\Pi_1 = (\{a, b, p\}, e, A)$ be an Automaton-like P colony with one agent $A = (ee, P, \{pp\})$. Let the programs of the agent be the following:

1. $\langle e \leftrightarrow a; e \leftrightarrow b \rangle$
2. $\langle a \rightarrow p; b \rightarrow p \rangle$

Let (ee, w), $w \in \{a, b\}^*$ be the initial configuration of Π_1. There is only one applicable program in this configuration, namely, program 1. This program is applicable only if ab is substring of w. Let $w = u{\cdot}ab{\cdot}v$, $u, v \in \{a, b\}^*$. After the first step of the computation the configuration of Π_1 is (ab, uv). In this configuration the second program $\langle a \rightarrow p; b \rightarrow p \rangle$ is applicable and the APCol system enters configuration (pp, uv). Because of presence of two copies of object p in the state of the agent, there is no applicable program in the APCol system and the computation halts.

The language accepted by Π_1 is $L(\Pi_1) = \{ab\}$.

Example 2. Let $\Pi_2 = (\{a, b, q\}, e, B)$ be an Automaton-like P colony with one agent $B = (ee, P, \{qe\})$. The programs of the agent are following:

1. $\langle e \leftrightarrow a; e \leftrightarrow b \rangle$
2. $\langle a \rightarrow q; b \rightarrow e \rangle$
3. $\langle q \rightarrow q; e \leftrightarrow a \rangle$
4. $\langle q \rightarrow q; e \leftrightarrow b \rangle$
5. $\langle q \rightarrow q; a \rightarrow e \rangle$
6. $\langle q \rightarrow q; b \rightarrow e \rangle$

Let (ee, w), $w \in \{a, b\}^*$ be the initial configuration of Π_2. There is only one applicable program in this configuration, program 1. This program is applicable only if ab is substring of w. Let $w = u{\cdot}ab{\cdot}v$, $u, v \in \{a, b\}^*$. After the first step of the computation the configuration of Π_2 is (ab, uv). In this configuration the second program $\langle a \rightarrow q; b \rightarrow e \rangle$ is applicable and the APCol system enters configuration (qe, uv). Because of presence of one copy of object q inside the agent, it uses program 3 (or 4) and, consequently, program 5 (or 6). By use of these programs, the agent erases every object a and b from the string.

The language accepted by Π_2 is $L(\Pi_2) = \{u{\cdot}ab{\cdot}v \mid u, v \in \{a, b\}^*\}$.

By the following example, we present a very simple Automaton-like P colony with only one agent with two programs that accepts the context-free language $L = \{a^n b^n \mid n \geq 0\}$.

Example 3. Let $\Pi_3 = (\{a, b\}, e, B)$ be an Automaton-like P colony with one agent $A = (ee, P, \{ee\})$. The programs of the agent are the following:

1. $\langle e \leftrightarrow a; e \leftrightarrow b \rangle$

2. $\langle a \to e; b \to e \rangle$

Let (ee, w), $w \in \{a, b\}^*$ be the initial configuration of Π. There is only one applicable program in this configuration, program 1. This program is applicable only if ab is substring of w. Let $w = u \cdot ab \cdot v$, $u, v \in \{a, b\}^*$. After the first step of the computation the configuration of the APCol system is (ab, uv). In this configuration the second program $\langle a \to e; b \to e \rangle$ is applicable and Π_3 enters configuration (ee, uv). This means that in two step of computation agent erases substring ab from a string. The computation ends when no substring ab occurs in the input string, thus any accepted word must be of the form $a^n b^n, n \geq 0$.

An interesting question is if we can construct an Automaton-like P colony which accepts the language $L = \{a^n b^n c^n \mid n \geq 0\}$.

Example 4. Let $\Pi_4 = (\{a, b, B\}, e, A_1, A_2, \{((ee), (bB))\})$ be an Automaton-like P colony with two agents $A_1 = (eB, P_1, \{ee\})$ and $A_2 = (bB, P_2, \{bB\})$. The programs of the agents are the following:

Set of programs P_1:
1. $\langle e \leftrightarrow a; B \leftrightarrow b \rangle$
2. $\langle a \to e; b \to B \rangle$
3. $\langle a \to e; b \to e \rangle$
4. $\langle e \leftrightarrow B; e \leftrightarrow c \rangle$
5. $\langle B \to e; c \to e \rangle$
6. $\langle e \leftrightarrow a; e \to F \rangle$
7. $\langle e \leftrightarrow b; e \to F \rangle$

Set of programs P_2:
A. $\langle b \leftrightarrow B; B \leftrightarrow b \rangle$

It can be shown that $L(\Pi_4) = \{a^n b^n c^n \mid n \geq 0\}$. Instead of the formal proof, we provide the sketch of the main idea. Π_4 functions as follows:

1. the first agent is an "eraser" - it erases substrings from the input string - it replaces string ab by B and erases substring Bc.
2. The second agent moves object B over that part of the input string which contains objects b.
3. The computation is accepting if every object from the input string is erased and the state of the first agent consists of only objects e.

We show an example of an accepting computation of Π_4 over the string $w = aaabbbccc$.

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aaabbbccc$	eB	1.	Bb	
1.	$aaBbbccc$	ab	**2.**, 3.	Bb	$A.$
2.	$aabBbccc$	eB	1.	Bb	$A.$
3.	$aBbBccc$	ab	**2.**, 3.	Bb	$A.$
4.	$abBBccc$	eB	1.	Bb	
5.	$BBBccc$	ab	2., **3.**	Bb	
6.	$BBBccc$	ee	4.	Bb	
7.	$BBcc$	Bc	5.	Bb	
8.	$BBcc$	ee	4.	Bb	
9	Bc	Bc	5.	Bb	
10.	Bc	ee	4.	Bb	
11.	ε	Bc	5.	Bb	
12.	ε	ee		Bb	

The next example is a non-accepting computation over the string $w = aaabbbccc$ on the next table.

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aaabbbccc$	eB	1.	Bb	
1.	$aaBbbccc$	ab	**2.**, 3.	Bb	$A.$
2.	$aabBbccc$	eB	1.	Bb	$A.$
3.	$aBbBccc$	ab	2., **3.**	Bb	$A.$
4.	$abBBccc$	ee	**6.**, 7.	Bb	
5.	$bBBccc$	Fe		Bb	

The Automaton-like P colony processes the string $u = aabcbc$ in the following computations.

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aabcbc$	eB	1.	Bb	
1.	$aBcbc$	ab	**2.**, 3.	Bb	$A.$
2.	$aBcbc$	eB		Bb	

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aabcbc$	eB	1.	Bb	
1.	$aBcbc$	ab	2., **3.**	Bb	
2.	$aBcbc$	ee	4.	Bb	
3.	abc	Bc	5.	Bb	
4.	abc	ee	**6.**, 7.	Bb	
5.	bc	Fa		Bb	

By the previous examples we obtain the following corollary.

Corollary 1. *The family of languages accepted by Automaton-like P colonies with one agent properly includes the family of languages accepted by jumping finite automata.*

Automaton-like P colonies with one agent are able to simulate jumping finite automata. If we add one more agent, we can construct Automaton-like P colonies simulating the work of two-counter machines. For details on two-counter machines consult [11, 16].

A two-counter machine is a 3-tape Turing machine $M = (\Sigma \cup \{Z, B\}, Q, R)$ where:

- Σ is an finite set of symbols - alphabet of input symbols,
- Z is the symbol marking the beginning of second and third tape (first and second counter),
- B is the blank symbol,
- Q is a finite set of states with two distinguished elements $q_0, q_f \in Q$, q_0 is the initial state and q_f is the final state of M,
- R is a set of transition rules.

The first tape of the machine is the input tape and second and third tapes are called storage tapes. All tapes are read-only and the storage tapes are semi-infinite. At the beginning of the storage tape symbol Z is found and the rest of the tape is empty (it contains only symbols B). The input tape contains the input word and occurrences of B.

The transition rules have the form $x = \langle b, q, c_1, c_2, q', e_1, e_2, g \rangle$ where:

- $b \in \Sigma \cup \{B\}$ is the symbol scanned by the input head on the input tape,
- $q, q' \in Q$ is the state of the two-counter machine,
- $c_1, c_2 \in \{Z, B\}$ are the symbols scanned on the storage tapes,
- $e_1, e_2 \in \{-1, 0, +1\}$ describe the move of the heads on the storage tapes,
- $g \in \{0, +1\}$ describe the move of the input head.

This rule can be performed by M if symbol scanned on input tape is b, M is in the state q and symbols read from storage tapes are c_1, c_2. By applying this rule M enters state q' and the heads move in according to g, e_1, e_2. If the value is -1, then the head moves to the left, if the value is $+1$, then the head moves to the right, and, if its value is 0, then the head stays at the same position. The head on

input tape can never move to the left and if symbol on read from storage tape is Z, then the head must not to move to the left, too.

The integer stored in a counter is number of all blank symbols laying between the storage head (and under it) and symbol Z on the storage tape. If the head scans symbol Z, then the counter has value 0.

The state of M, the contents of the tapes, and the position of heads on the tapes together define configuration of M. The two-counter machine is in an initial configuration if it is in initial state, on the first tape it has the input word, on the second and third tape it has a word from ZB^*, and the heads are scanning the first symbol of the tapes. The two-counter machine M is in the accepting configuration if the input head reads the last non-blank symbol on the input tape and the machine is in the final state; in this case the input word is accepted.

Two-counter machine are computationally complete computing devices [11, 8].

Theorem 2. *Let Σ be an alphabet, $L \subseteq \Sigma^*$ be a recursively enumerable language. Let $L' = S \cdot L \cdot E$, where $S, E \notin \Sigma$. Then there exists an Automaton-like P colony Π with two agents such that $L' = L(\Pi)$ holds.*

Proof. Let $M = (\Sigma \cup \{Z, B\}, Q, R)$ be a two-counter machine accepting language L. We construct an Automaton-like P colony $\Pi = (O, e, A_1, A_2)$ which accepts $S \cdot L \cdot E$, where:

- $O = \Sigma \cup \{e, Z_1, Z_2, B_1, B_2, D, G, E, S, S', \overline{S}, \overline{\overline{S}}, Q, \overline{Q}, \overline{\overline{Q}}, M, \overline{M}, \overline{\overline{M}}, \} \cup$
  $\cup \{ {}^{A}_{B} \mid A \in \Sigma \cup \{Z_1, Z_2, B_1, B_2\},$
  $B \in \{T, T_1, T_2, H_{+1}, H_0, I_{+1}, I_{-1}, I_0, J_{+1}, J_{-1}, J_0\} \cup$
  $\cup \{q, \overline{q}, \overline{\overline{q}}, \overline{\overline{\overline{q}}} \mid q \in Q\}\}$
- $A_1 = (S'G, P_1, \{\overline{\overline{M}}\, {}^{A}_{H},$ where $A \in \Sigma \cup \{e\}, H \in \{H_{+1}, H_0\}\})$
- $A_2 = ({}^{Z_1\ Z_2}_{T_1\ T_2}, P_2, \{\overline{\overline{Q}}e\})$

The sets of programs are the following:

For the first part of computation, programs are needed to initialize the simulation. They generate the initial contents of the counters and mark the first symbols on each tape, i.e., the symbols under the reading heads (for example, replace a by ${}^{a}_{T}$).

The programs for initialization of the simulation:

A_1 :	A_2 :
1. $\langle S' \leftrightarrow S; G \leftrightarrow a \rangle$	1. $\left\langle {}^{Z_1}_{T_1} \leftrightarrow e; {}^{Z_2}_{T_2} \leftrightarrow S' \right\rangle$
2. $\left\langle S \rightarrow \overline{S}; a \rightarrow {}^{a}_{T} \right\rangle$	2. $\langle S' \rightarrow D; e \rightarrow e \rangle$
3. $\left\langle \overline{S} \rightarrow \overline{\overline{S}}; {}^{a}_{T} \leftrightarrow G \right\rangle$	
4. $\left\langle \overline{\overline{S}} \rightarrow q_0; G \rightarrow {}^{a}_{H} \right\rangle$	
$\forall a \in \Sigma,\ H \in \{H_{+1}, H_0\}$	

Let the input word of the two-counter machine be $w = av,\ a \in \Sigma,\ w, v \in \Sigma^*$. At the beginning of the computation the input tape contains the word $SavE$.

Agent A_1 uses the program 1. and replaces first two symbols Sa by string $S'G$. In the second step, both agents work. Agent A_1 rewrites objects Sa to objects $\overline{S}\,{}^{a}_{T}$ (inside it), and the second agent A_2 replaces symbol S' by string ${}^{Z_1\ Z_2}_{T_1\ T_2}$. In the third step agent A_1 replaces symbol G by symbol ${}^{a}_{T}$ and rewrites object $\overline{S}$ by object $\overline{\overline{S}}$. Agent A_2 changes its state from $(S'e)$ to De. Agent A_2 is prepared to continue the simulation, otherwise agent A_1 has to do one more step. At the last step of the initialization, agent A_1 uses program 4. and it rewrites $\overline{\overline{S}}G$ to $q_0\,{}^{a}_{H}$.

$$(SavE; S'G, {}^{Z_1\ Z_2}_{T_1\ T_2}) \Rightarrow (S'GvE; Sa, {}^{Z_1\ Z_2}_{T_1\ T_2}) \Rightarrow ({}^{Z_1\ Z_2}_{T_1\ T_2}GvE; \overline{S}\,{}^{a}_{T}, S'e) \Rightarrow$$
$$\Rightarrow ({}^{Z_1\ Z_2\ a}_{T_1\ T_2\ T}vE; \overline{\overline{S}}G, De) \Rightarrow ({}^{Z_1\ Z_2\ a}_{T_1\ T_2\ T}vE; q_0\,{}^{a}_{H}, De)$$

The second group of programs is to simulate the execution of the transition rules of the two-counter machine. Let $\langle b, q, c_1, c_2, q', e_1, e_2, g\rangle$ be a transition rule, where $b \in \Sigma \cup \{B\}$, $q, q' \in Q$, $c_1, c_2 \in \{Z, B\}$, $e_1, e_2 \in \{-1, 0, +1\}$, $g \in \{0, +1\}$. Agent A_1 has programs to check whether the reading heads are over the corresponding symbols, i.e., symbols b, c_1 and c_2, and to note the string moves of the reading heads by replacing ${}^{b}_{T}$, ${}^{c_1}_{T_1}$ and ${}^{c_2}_{T_2}$ by ${}^{b}_{g}$, ${}^{c_1}_{e_1}$ and ${}^{c_2}_{e_2}$.

A_1 :

5. $\left\langle q \to \overline{q}; {}^{b}_{g} \leftrightarrow {}^{b}_{T} \right\rangle$
6. $\left\langle \overline{q} \to \overline{q_1}; {}^{b}_{T} \to {}^{c_{1\,1}}_{I_{e_1}} \right\rangle$
7. $\left\langle \overline{q_i} \to \overline{q_{i+1}}; {}^{c_{1\,1}}_{I_{e_1}} \to {}^{c_{1\,1}}_{I_{e_1}} \right\rangle; 1 \geq i \geq 5$
8. $\left\langle \overline{q_6} \to \overline{\overline{q}}; {}^{c_{1\,1}}_{I_{e_1}} \leftrightarrow {}^{c_{1\,1}}_{T_1} \right\rangle$
9. $\left\langle \overline{\overline{q}} \to \overline{\overline{q_1}}; {}^{c_{1\,1}}_{T_1} \to {}^{c_{2\,2}}_{J_{e_2}} \right\rangle$
10. $\left\langle \overline{\overline{q_j}} \to \overline{\overline{q_{j+1}}}; {}^{c_{2\,2}}_{J_{e_2}} \to {}^{c_{2\,2}}_{J_{e_2}} \right\rangle; j \in \{1, 2\}$
11. $\left\langle \overline{\overline{q_3}} \to \overline{\overline{q}}; {}^{c_{2\,2}}_{J_{e_2}} \leftrightarrow {}^{c_{2\,2}}_{T_2} \right\rangle$
12. $\left\langle \overline{\overline{q}} \to \overline{\overline{q_1}}; {}^{c_2}_{T_2} \to {}^{a}_{H} \right\rangle$
13. $\left\langle \overline{\overline{q_k}} \to \overline{\overline{q_{k+1}}}; {}^{a}_{H} \to {}^{a}_{H} \right\rangle; k \in \{1, 2\}$
14. $\left\langle \overline{\overline{q_3}} \to q'; {}^{a}_{H} \to {}^{a}_{H} \right\rangle$

$\forall a \in \Sigma \cup \{B\},\ H \in \{H_{+1}, H_0\}$

Agent A_1 performs some "waiting" steps to let the second agent execute the movement of reading heads. At the last steps of this part of the computation agent A_1 in some way "precomputes" the symbol to be read in the next step. This symbol is non-deterministically chosen from the set $\Sigma \cup \{B\}$. If the precomputed symbol does not match the symbol on the tape in the next step, the computation halts in a non-accepting configuration. The second agent has programs to perform the movement of the reading heads.

A_2 :

the movement of reading head on the input tape

15. $\left\langle D \leftrightarrow {}^{b}_{H_0}; e \rightarrow {}^{b}_{T} \right\rangle$	17. $\left\langle D \leftrightarrow {}^{b}_{H_{+1}}; e \leftrightarrow a \right\rangle$	19. $\left\langle D \leftrightarrow {}^{b}_{H_{+1}}; e \leftrightarrow E \right\rangle$
16. $\left\langle {}^{b}_{H_0} \rightarrow e; {}^{b}_{T} \leftrightarrow D \right\rangle$	18. $\left\langle {}^{b}_{H_{+1}} \rightarrow b; a \rightarrow {}^{a}_{T} \right\rangle$	20. $\left\langle {}^{b}_{H_{+1}} \rightarrow b; E \rightarrow B' \right\rangle$
		21. $\langle b \leftrightarrow D; B' \leftrightarrow e \rangle$
		22. $\left\langle D \leftrightarrow B'; e \rightarrow {}^{B}_{T} \right\rangle$
		23. $\left\langle B' \rightarrow E; {}^{B}_{T} \rightarrow {}^{B}_{T} \right\rangle$
		24. $\left\langle {}^{B}_{T} \leftrightarrow D; E \leftrightarrow e \right\rangle$

$\forall a \in \Sigma,\ H \in \{H_{+1}, H_0\}$

In the first column there are programs for the case when the reading head does not move - $g = 0$. In the second and third column there are programs for executing the move of the head to the right - $g = +1$. In the third column there are programs to do movement to the right when the reading head is on the last symbol of the string, so the head have to move to the blank symbol after the string.

A_2 :

the movement of reading head on the first counter, $X \in \{Z_1, B_1\}$

25. $\left\langle D \leftrightarrow {}^{X}_{I_0}; e \rightarrow {}^{X}_{T_1} \right\rangle$	27. $\left\langle D \leftrightarrow {}^{X}_{I_{+1}}; e \rightarrow {}^{B_1}_{T_1} \right\rangle$	30. $\left\langle D \leftrightarrow Y; e \leftrightarrow {}^{X}_{I_{-1}} \right\rangle$
26. $\left\langle {}^{X}_{I_0} \rightarrow e; {}^{X}_{T_1} \leftrightarrow D \right\rangle$	28. $\left\langle {}^{X}_{I_{+1}} \rightarrow X; {}^{B_1}_{T_1} \rightarrow {}^{B_1}_{T_1} \right\rangle$	31. $\left\langle Y \rightarrow {}^{Y}_{T_1}; {}^{X}_{I_{-1}} \rightarrow e \right\rangle$
	29. $\left\langle X \leftrightarrow D; {}^{B_1}_{T_1} \leftrightarrow e \right\rangle$	28. $\left\langle {}^{Y}_{T_1} \leftrightarrow D; e \leftrightarrow e \right\rangle$

A_2 :

the movement of reading head on the second counter, $X \in \{Z_2, B_2\}$

32. $\left\langle D \leftrightarrow {}^{X}_{J_0}; e \rightarrow {}^{X}_{T_2} \right\rangle$	34. $\left\langle D \leftrightarrow {}^{X}_{J_{+1}}; e \rightarrow {}^{B_2}_{T_2} \right\rangle$	37. $\left\langle D \leftrightarrow Y; e \leftrightarrow {}^{X}_{J_{-1}} \right\rangle$
33. $\left\langle {}^{X}_{J_0} \rightarrow e; {}^{X}_{T_2} \leftrightarrow D \right\rangle$	35. $\left\langle {}^{X}_{J_{+1}} \rightarrow X; {}^{B_2}_{T_2} \rightarrow {}^{B_2}_{T_2} \right\rangle$	38. $\left\langle Y \rightarrow {}^{Y}_{T_2}; {}^{X}_{J_{-1}} \rightarrow e \right\rangle$
	36. $\left\langle X \leftrightarrow D; {}^{B_2}_{T_2} \leftrightarrow e \right\rangle$	39. $\left\langle {}^{Y}_{T_2} \leftrightarrow D; e \leftrightarrow e \right\rangle$

The programs devoted to reading heads on the counters are similar to the ones for the reading head on the input tape. The programs in the first columns are for staying at the same place, the second columns for movement to the right, and the last columns for the movement of the reading head to the left.

Now we finish the simulation of execution of the transition rules. The last group of programs is to finish the computation after the two-counter machine M comes to final state. M accepts the string on the input tape only if it is in the final state and the reading head on the input tape is in the position on the last non-blank symbol. The Automaton-like P colony Π ends computation by halting. The string is accepted by Π only if the input tape is empty after halting in the final state of at least one agent. So, after reaching the final state of M the Automaton-like P colony Π has to erase the symbols on and before the reading head (symbol of

the type $\begin{smallmatrix} a \\ T \end{smallmatrix}$ and all symbols placed on the left from this symbol - the contents of both counters) on the input tape, symbols B and finally the symbol E, which determines the end of the string w. If there is some non-erased symbol left on the input tape, then the two-counter machine does not accept the word w, too, because the reading head in not on the last non-blank symbol of the input tape.

A_1 :

erasing the last symbol over the reading head on the input tape
$a \in \Sigma, H \in H_0, H_{+1}$

32. $\left\langle q_f \leftrightarrow M; \begin{smallmatrix} a \\ H \end{smallmatrix} \leftrightarrow \begin{smallmatrix} a \\ T \end{smallmatrix} \right\rangle$ 34. $\left\langle \overline{M} \rightarrow \overline{\overline{M}}; Q \leftrightarrow \begin{smallmatrix} a \\ H \end{smallmatrix} \right\rangle$
33. $\left\langle M \rightarrow \overline{M}; \begin{smallmatrix} a \\ T \end{smallmatrix} \rightarrow Q \right\rangle$ 35. $\left\langle \overline{\overline{M}} \rightarrow \overline{\overline{M}}; \begin{smallmatrix} a \\ H \end{smallmatrix} \rightarrow e \right\rangle$

A_2 :

erasing symbols on the counters and at the end of the tape
$Z \in \{Z_1, Z_2, B_1, B_2, \begin{smallmatrix} Z_1 \\ T_1 \end{smallmatrix}, \begin{smallmatrix} Z_2 \\ T_2 \end{smallmatrix}, \begin{smallmatrix} B_1 \\ T_1 \end{smallmatrix}, \begin{smallmatrix} B_2 \\ T_2 \end{smallmatrix}, B, E\}$

36. $\left\langle D \rightarrow \overline{Q}; e \rightarrow Q \right\rangle$ 38. $\left\langle \overline{\overline{Q}} \leftrightarrow \overline{\overline{q}}; e \leftrightarrow Z \right\rangle$
37. $\left\langle \overline{Q} \rightarrow \overline{\overline{Q}}; Q \leftrightarrow e \right\rangle$ 39. $\left\langle \overline{\overline{Q}} \rightarrow \overline{\overline{Q}}; Z \rightarrow e \right\rangle$

The computation of Π starts in the initial configuration which corresponds to the initial configuration of two-counter machine M. After the initialization, the simulation of execution of particular transition rules runs in the same way as they are applied by M. The computation of Π halts in accepting configuration only if M processes the whole input string and ends computation in the one of final states. □

By the previous theorem we obtain the following corollary:

Corollary 2. *Any recursively enumerable language can be obtained as a projection of a language accepted by an Automaton-like P colony with two agents.*

4 Conclusions

We introduced the concept on an Automata-like P colony (an APCol system) - a variant of P colonies that works on a string. The agents communicate with the environment alike standard P colonies: they process symbols in the string. As P colony automata, the concept is a notion combining properties of P colonies and classical automata. The main difference in the two notions is in the way of interaction between the agents (the cells) and the environment. We compared the computational power of Automata-like P colonies and that of jumping finite automata, and proved that APCol systems are strictly powerful than jumping finite automata. We also provided a representation of the recursively enumerable language class in terms of APCol systems. The question of exact description of the computational power of Automata-like P colonies is still open.

Remark 1. This work was partially supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/-02.0070), by SGS/24/2011 and by project OPVK no. CZ.1.07/2.2.00/28.0014. and in part by the Hungarian Scientific Research Fund, "OTKA", project K75952.

References

1. L. Ciencialová, L. Cienciala, Variation on the theme: P colonies. In: Proc. 1st Intern. Workshop on Formal Models. (D. Kolář, A. Meduna, eds.), Ostrava, 2006, 27–34.
2. L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, Gy. Vaszil, Variants of P colonies with very simple cell structure. International Journal of Computers, Communication and Control 4(3) (2009), 224–233.
3. L. Ciencala, L. Ciencialová, E. Csuhaj-Varjú, Gy. Vaszil, PCol Automata: Recognizing strings with P colonies. In: Proc. BWMC 2010, Sevilla, 2010, Ed. by M. A. Martnez-del-Amor et al. Fnix Editora, Sevilla, 2010, 65-76.
4. L. Cienciala, L. Ciencialová, A. Kelemenová, Homogeneous P colonies. Computing and Informatics 27 (2008), 481–496.
5. L. Cienciala, L. Ciencialová, A. Kelemenová, On the number of agents in P colonies. In: Membrane Computing. 8th International Workshop, WMC 2007. Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers. (G. Eleftherakis et. al, eds.), LNCS 4860, Springer-Verlag, Berlin-Heidelberg, 2007, 193–208.
6. 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.
7. E. Csuhaj-Varjú, M. Margenstern, Gy. Vaszil, P colonies with a bounded number of cells and programs. In: Membrane Computing. 7th International Worskhop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006. Revised, Selected and Invited Papers. (H-J. Hoogeboom et. al, eds), LNCS 4361, Springer-Verlag, Berlin-Heidelberg, (2007), 352–366.
8. P. C. Fischer, Turing machines with restricted memory access. Information and Control, 9, 364–379, 1966.
9. R. Freund, M. Oswald, P colonies working in the maximally parallel and in the sequential mode. Pre-Proc. In: 1st Intern. Workshop on Theory and Application of P Systems. (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, 49–56.
10. R. Freund, M. Oswald, P colonies and prescribed teams. International Journal of Computer Mathematics 83 (2006), 569–592.
11. Hopcroft, J.E., Ullman, J.D., Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, Mass., 1979.
12. J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multi-agent systems. Cybernetics and Systems 23 (1992), 621–633.
13. J. Kelemen, A. Kelemenová, Gh. Păun, Preview of P colonies: A biochemically inspired computing model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX). (M. Bedau et al., eds.), Boston Mass., 2004, 82–86.
14. A. Kelemenová, P Colonies. Chapter 23.1, In: The Oxford Handbook of Membrane Computing. (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2010, 584–593.

15. A. Meduna, P. Zemek, Jumping Finite Automata. Int. J. Found. Comput. Sci. 23, 2012, pp. 1555–1578.
16. M. Minsky, Computation – Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967.
17. Gh. Păun, G. Rozenberg, A. Salomaa, eds.,The Oxford Handbook of Membrane Computing. Oxford University Press, 2010.

Scalable Grid-Based Implementation for Membrane Computing

Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
gabriel@info.uaic.ro

Summary. We first present the formal semantics of a parallel rule-based formalism inspired by biological cells, and then provide a faithful parallel implementation of this computational model by using GridGain and taking care of various synchronization issues. Synchronization is achieved by using barriers and preconditions; both refer to the fact that a membrane can apply its rules only after it has received signals from the other related membranes. We develop a scalable parallel implementation using the MapReduce paradigm in GridGain which allows the splitting of a task into multiple subtasks, the parallel execution of these subtasks in parallel and the aggregation of the partial results into a single, final result. This implementation is very close to the formal description of this parallel model of membrane systems, a model which is computationally equivalent to Turing machines and able to provide polynomial solutions to NP-complete problems.

1 Introduction

Membrane systems are essentially parallel and nondeterministic computing models inspired by the compartments of (eukaryotic) cells and their biochemical reactions. The structure of a cell is represented by a set of hierarchically embedded membranes, all of which are contained inside a skin membrane. The molecular species (ions, proteins, etc.) floating inside cellular compartments are represented by multisets of objects described by means of symbols over a given alphabet. Objects can be modified or communicated between adjacent compartments. Chemical reactions are represented by evolution rules which operate on the objects, as well as on the compartmentalized structure (by dissolving, dividing, creating, or moving membranes).

Membrane systems (also called P systems) perform parallel computations in the following way: starting from an initial configuration (the initial membrane structure and the initial multisets of objects placed inside the membranes), a system evolves by applying the evolution rules of each membrane in a nondeterministic manner. A rule is applicable when all the objects which appear in its left hand side are available in the membrane where the rule is placed.

Since membrane systems aim to abstract computing ideas and models from the structure and the functioning of living cells, several extensions come from both biology (aiming to model more and more biological phenomena) and computer science (aiming to add new computing features). Their computing power and efficiency have been investigated using the approaches of formal languages, grammars, register machines and complexity theory. Membrane systems are presented together with many variants and examples in [7]. Several applications of these systems are presented in [5]. An updated bibliography can be found at the P systems web page http://ppage.psystems.eu. The state of the art is presented in the handbook published recently by Oxford University Press [8].

In this paper we present a paralel implementation of membrane systems by using GridGain [9], using a new appealing technology. The implementation is derived after studying some synchronization issues in membrane systems by defining their operational semantics and describing the parallel (sub)steps of their evolutions.

2 Operational Semantics of the Membrane Systems

The basic model of membrane computing is usually referred to as a transition membrane systems In this model, objects are represented using symbols from a given alphabet, and each symbol from this alphabet can appear inside a region in many different copies. A membrane system is composed of membranes which do not intersect, and which are all contained within a skin membrane. Each membrane can contain multisets of objects, evolution rules and other membranes. The objects inside a membrane evolve in a maximal parallel manner according to the evolution rules inside the same membrane. According to [7], maximal parallel "means that we assign objects to rules, non-deterministically choosing the objects and the rules, until no further assignment is possible."

First we present an abstract syntax for membrane systems, and then we a structural operational semantics of these systems by means of three sets of inference rules corresponding to maximal parallel rewriting, parallel communication, and parallel dissolving.

In general, operational semantics provide a way of rigorously describing the evolution of a computing system. Configurations are states of a transition system, and a computation consists of a sequence of transitions from one configuration to another, until a final configuration is reached (if the computation terminates, that is). Structural operational semantics provides a framework for defining a formal description of a computing system. In basic membrane systems, a computation is regarded as a sequence of parallel applications of rules in various membranes, followed by a communication step and a dissolving step. A structural operational semantics of membrane systems emphasizes the deductive nature of membrane computing by describing the transition steps through a set of inference rules. Considering a set $\mathcal{R}$ of inference rules of the form $\frac{premises}{conclusion}$, the evolution of a membrane system can be presented as a deduction tree.

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

2.1 Configurations and Transitions

First we present an inductive definition of the membrane structure, the sets of configurations for a membrane system, and an intuitive definition for the transition systems, which is given by considering each transition step: maximal parallel rewriting, parallel communication, and parallel dissolving.

Let O be a finite alphabet of objects over which we consider the *free commutative monoid* O_c^*, whose elements are *multisets*. The empty multiset is denoted by *empty*. Objects can be enclosed in messages together with a target indication. We have *here* messages of typical form $(w, here)$, *out* messages (w, out), and *in* messages (w, in_L). For the sake of simplicity, hereinafter we consider that the messages with the same target indication merge into one message: $\prod_{i\in I}(v_i, here) = (w, here)$, $\prod_{i\in I}(v_i, in_L) = (w, in_L)$, $\prod_{i\in I}(v_i, out) = (w, out)$, with $w = \prod_{i\in I} v_i$, I a non-empty set, and $(v_i)_{i\in I}$ a family of multisets over O.

We use the mappings rules and $\mathsf{priority}$ to associate to a membrane label the set of evolution rules and the priority relation over rules (when this exists) : $\mathsf{rules}(L_i) = R_i$, $\mathsf{priority}(L_i) = \rho_i$, and the projections L and w which return from a membrane its label and its current multiset, respectively.

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

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

We conventionally suppose the existence of a set of sibling membranes denoted by $NULL$ such that $M, NULL = M = NULL, M$ and $\langle\, L \,|\, w \,;\, NULL \,\rangle = \langle\, L \,|\, w \,\rangle$. The use of $NULL$ significantly simplifies several definitions and proofs. Let $\mathcal{M}^*(\Pi)$ be the free commutative monoid generated by $\mathcal{M}(\Pi)$ with the operation $(\_, \_)$ and the identity element $NULL$. We define $\mathcal{M}^+(\Pi)$ as the set of elements from $\mathcal{M}^*(\Pi)$ without the identity element. Let M_+, N_+ range over non-empty sets of sibling membranes, M_i over membranes, M_*, N_* range over possibly empty multisets of sibling membranes, and L over labels. The membranes preserve the initial labeling, evolution rules and priority relation among them in all subsequent configurations.

Therefore in order to describe a membrane we consider its label and the current multiset of objects together with its structure.

A *configuration* for a P system Π is a skin membrane which has no messages and no dissolving symbol δ, i.e., the multisets of all regions are elements in O_c^*. We denote by $\mathcal{C}(\Pi)$ the set of configurations for Π.

An *intermediate configuration* is an arbitrary skin membrane in which we may find messages or the dissolving symbol δ. We denote by $\mathcal{C}^{\#}(\Pi)$ the set of intermediate configurations. We have $\mathcal{C}(\Pi) \subseteq \mathcal{C}^{\#}(\Pi)$.

Each membrane system has an initial configuration which is characterized by the initial multiset of objects for each membrane and the initial membrane structure of the system. For two configurations C_1 and C_2 of Π, we say that there is a *transition* from C_1 to C_2, and write $C_1 \Rightarrow C_2$, if the following *steps* are executed in the given order:

1. *maximal parallel rewriting step*: each membrane evolves in a maximal parallel manner;
2. *parallel communication of objects through membranes* by sending and receiving messages;
3. *parallel membrane dissolving*, consisting in dissolving the membranes containing δ.

The last two steps take place only if there are messages or δ symbols resulting from the first step, respectively. If the first step is not possible, then neither are the other two steps; we say that the system has reached a *halting configuration*.

2.2 Maximal Parallel Rewriting Step

We briefly present an operational semantics for membrane systems, considering each of the three steps. First we formally define the maximal parallel rewriting $\stackrel{mpr}{\Longrightarrow}_L$ for a multiset of objects in one membrane, and we extend it to maximal parallel rewriting $\stackrel{mpr}{\Longrightarrow}$ over several membranes. Some preliminary notions are required.

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

- *a multiset of messages and the dissolving symbol δ are* **L-irreducible***;*
- *a multiset of objects w is* **L-irreducible** *iff there are no rules in* $\mathsf{rules}(L)$ *applicable to w with respect to the priority relation* $\mathsf{priority}(L)$*;*
- *a simple membrane $\langle\, L \mid w \,\rangle$ is* **mpr-irreducible** *iff w is L-irreducible;*
- *a non-empty set of sibling membranes $M_1, \ldots, M_n$ is* **mpr-irreducible** *iff M_i is mpr-irreducible for every $i \in [n]$; $NULL$ is* **mpr-irreducible***;*
- *a composite membrane $\langle\, L \mid w\ ;\ M_1, \ldots, M_n \,\rangle$ is* **mpr-irreducible** *iff w is L-irreducible, and the set of sibling membranes $M_1, \ldots, M_n$ is mpr-irreducible.*

The priority relation is a form of control on the application of rules. In the presence of a priority relation, no rule of a lower priority can be used during the same evolution step when a rule with a higher priority is used, even if the two rules do not compete for the same objects. We formalize the conditions imposed by the priority relation on rule applications in the definition below.

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

- $R \subseteq \mathsf{rules}(L)$,
- $w = u_1 \ldots u_n z$, *so each rule* $r \in R$ *is applicable on* w,
- $(\forall r \in R, \forall r' \in \mathsf{rules}(L))$ r' *applicable on* w *implies* $(r', r) \notin \mathsf{priority}(L)$ *(we have* $(r_1, r_2) \in \mathsf{priority}(L)$ *iff* $r_1 > r_2$ *),*
- $(\forall r', r'' \in R)$ $(r', r'') \notin \mathsf{priority}(L)$,
- *the dissolving symbol* δ *has at most one occurrence in the multiset* $v_1 \ldots v_n$.

Maximal parallel rewriting relations $\overset{mpr}{\Longrightarrow}_L$ and $\overset{mpr}{\Longrightarrow}$ are defined by the following inference rules:

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

$$(\mathbf{R_1}) \ \frac{}{u_1 \ldots u_n z \overset{mpr}{\Longrightarrow}_L v_1 \ldots v_n z}$$

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

$$(\mathbf{R_2}) \ \frac{w \overset{mpr}{\Longrightarrow}_L w'}{\langle\, L \mid w \,;\, M_* \,\rangle \overset{mpr}{\Longrightarrow} \langle\, L \mid w' \,;\, M_* \,\rangle}$$

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

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

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

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

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

$$(\mathbf{R_5}) \ \frac{M \overset{mpr}{\Longrightarrow} M', M_+ \overset{mpr}{\Longrightarrow} M'_+}{M, M_+ \overset{mpr}{\Longrightarrow} M', M'_+}$$

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

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

We note that $\overset{mpr}{\Longrightarrow}$ for simple membranes can be described by rule $(\mathbf{R_2})$ with $M_* = NULL$.

Remark 1. M is mpr-irreducible iff there does not exist M' such that $M \overset{mpr}{\Longrightarrow} M'$.

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

The formal definition of $\overset{mpr}{\Longrightarrow}$ given above corresponds to the intuitive description of maximal parallelism. The nondeterminism is given by the associativity and commutativity of the concatenation operation over objects used in $\mathbf{R_1}$. The parallelism of the evolution rules in a membrane is also given by $\mathbf{R_1}$: $u_1 \dots u_n z \overset{mpr}{\Longrightarrow}_L v_1 \dots v_n z$ says that the rules of the multiset $(u_1 \to v_1, \dots, u_n \to v_n)$ are applied simultaneously. The fact that the membranes evolve in parallel is described by rules $\mathbf{R_3}-\mathbf{R_6}$.

2.3 Parallel Communication of Objects

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

$\mathsf{obj}(w)$ is obtained from w by removing all messages,

$$\mathsf{here}(w) = \begin{cases} empty & \text{if } w \text{ is } here\text{-free,} \\ w'' & \text{if } w = w'(w'', here) \wedge w' \text{ is } here\text{-free;} \end{cases}$$

$$\mathsf{out}(w) = \begin{cases} empty & \text{if } w \text{ is } out\text{-free,} \\ w'' & \text{if } w = w'(w'', out) \wedge w' \text{ is } out\text{-free;} \end{cases}$$

$$\mathsf{in}_L(w) = \begin{cases} empty & \text{if } w \text{ is } in_L\text{-free,} \\ w'' & \text{if } w = w'(w'', in_L) \wedge w' \text{ is } in_L\text{-free.} \end{cases}$$

We consider the extension of the operator w (previously defined over membranes) to non-empty sets of sibling membranes by setting $\mathsf{w}(NULL) = empty$ and $\mathsf{w}(M_1, \dots, M_n) = \mathsf{w}(M_1) \dots \mathsf{w}(M_n)$.

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

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

- *a simple membrane $\langle\ L \mid w\ \rangle$ is* **tar-irreducible** *iff w is here-free and $L \neq Skin \vee (L{=}Skin \wedge w$ out-free);*
- *a non-empty set of sibling membranes $M_1, \dots, M_n$ is* **tar-irreducible** *iff M_i is tar-irreducible for every $i \in [n]$; $NULL$ is* **tar-irreducible***;*

- *a composite membrane* $\langle\, L \mid w\,;\, M_1, \ldots, M_n \,\rangle$, $n \geq 1$, *is* **tar-irreducible** *iff:* w *is here-free and* $in_{L(M_i)}$*-free for every* $i \in [n]$, $L \neq Skin \vee (L = Skin \wedge w$ *is out-free),* $\mathsf{w}(M_i)$ *is out-free for all* $i \in [n]$*, and the set of sibling membranes* $M_1, \ldots, M_n$ *is tar-irreducible;*

Notation. We treat messages of the form $(w', here)$ as a particular communication inside a membrane, and we substitute $(w', here)$ by w'. We denote by $\overline{w}$ the multiset obtained by replacing $(\mathsf{here}(w), here)$ with $\mathsf{here}(w)$ in w. For instance, if $w = a\,(bc, here)\,(d, out)$ then $\overline{w} = abc\,(d, out)$, where $\mathsf{here}(w) = bc$. We note that $\mathsf{in}_L(\overline{w}) = \mathsf{in}_L(w)$, and $\mathsf{out}(\overline{w}) = \mathsf{out}(w)$.

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

For each tar-irreducible $M_* \in \mathcal{M}^*(\Pi)$ and multiset w such that $\mathsf{here}(w) \neq empty$, or $L = Skin \wedge \mathsf{out}(w) \neq empty$, or there exists $M_i \in M_*$ with $\mathsf{in}_{L(M_i)}(w)\mathsf{out}(\mathsf{w}(M_i)) \neq empty$,

$$(\mathbf{C_1})\frac{}{\langle\, L \mid w\,;\, M_* \,\rangle \stackrel{tar}{\Longrightarrow} \langle\, L \mid w'\,;\, M'_* \,\rangle}$$

where

$$w' = \begin{cases} \mathsf{obj}(\overline{w})\,\mathsf{out}(\mathsf{w}(M_*)) & \text{if } L = Skin, \\ \mathsf{obj}(\overline{w})\,(\mathsf{out}(w), out)\,\mathsf{out}(\mathsf{w}(M_*)) & \text{otherwise;} \end{cases}$$

and

$$\mathsf{w}(M'_i) = \mathsf{obj}(\mathsf{w}(M'_i))\,\mathsf{in}_{\mathsf{L}(M_i)}(w), \text{ for all } M_i \in M_*$$

For each $M_1, \ldots, M_n, M'_1, \ldots, M'_n \in \mathcal{M}^+(\Pi)$, and multiset w,

$$(\mathbf{C_2})\frac{M_1, \ldots, M_n \stackrel{tar}{\Longrightarrow} M'_1, \ldots, M'_n}{\langle\, L \mid w\,;\, M_1, \ldots, M_n \,\rangle \stackrel{tar}{\Longrightarrow} \langle\, L \mid w''\,;\, M''_1, \ldots, M''_n \,\rangle}$$

where

$$w'' = \begin{cases} \mathsf{obj}(\overline{w})\,\mathsf{out}(\mathsf{w}(M'_1, \ldots, M'_n)) & \text{if } L = Skin, \\ \mathsf{obj}(\overline{w})\,(\mathsf{out}(w), out)\,\mathsf{out}(\mathsf{w}(M'_1, \ldots, M'_n)) & \\ \text{otherwise;} & \end{cases}$$

and each M''_i is obtained from M'_i by replacing its resources with

$$\mathsf{w}(M''_i) = \mathsf{obj}(\overline{\mathsf{w}(M'_i)})\,\mathsf{in}_{\mathsf{L}(M'_i)}(w), \text{ for all } i \in [n]$$

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

$$(\mathbf{C_3})\frac{M \stackrel{tar}{\Longrightarrow} M'}{M, M_+ \stackrel{tar}{\Longrightarrow} M', M_+}$$

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

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

Remark 2. M is tar-irreducible iff there does not exist M' such that $M \stackrel{tar}{\Longrightarrow} M'$.

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

2.4 Parallel Membrane Dissolving

If the special symbol δ occurs in the multiset of objects of a membrane labeled by L, that membrane is dissolved, its evolution rules and the associated priority relation are lost, and its contents (objects and membranes) is added to the contents of the surrounding membrane. We say that a multiset w is *δ-free* if it does not contain the special symbol δ.

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

- *a simple membrane is* **δ-irreducible** *iff it has no messages;*
- *a non-empty set of sibling membranes $M_1, \ldots, M_n$ is* **δ-irreducible** *iff every membrane M_i is δ-irreducible, for $1 \leq i \leq n$; $NULL$ is* **δ-irreducible***;*
- *a composite membrane $\langle\, L \mid w\, ;\, M_+ \,\rangle$ is* **δ-irreducible** *iff w has no messages, M_+ is δ-irreducible, and $w(M_+)$ is δ-free;*

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

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

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

For each $M_+ \in \mathcal{M}^+(\Pi)$, $M'_* \in \mathcal{M}^*(\Pi)$, δ-free multiset w_2, multisets w_1, w'_2, and labels L_1, L_2

$$(\mathbf{D_2})\frac{\langle\, L_2 \mid w_2\, ;\, M_+\,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_2 \mid w'_2\, ;\, M'_*\,\rangle}{\langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2\, ;\, M_+\,\rangle\,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w'_2\, ;\, M'_*\,\rangle\,\rangle}$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, $M'_* \in \mathcal{M}^*(\Pi)$, multisets w_1, w_2, w'_2, and labels L_1, L_2

$$(\mathbf{D_3})\frac{\langle\, L_2 \mid w_2\delta\, ;\, M_+\,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_2 \mid w'_2\delta\, ;\, M'_*\,\rangle}{\langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2\delta\, ;\, M_+\,\rangle\,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w'_2\, ;\, M'_*\,\rangle}$$

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

$$(\mathbf{D_4})\frac{\langle\, L \mid w\, ;\, M_+\,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w'\, ;\, M'_*\,\rangle}{\langle\, L \mid w\, ;\, M_+, N_+\,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w'\, ;\, M'_*, N_+\,\rangle}$$

$$(\mathbf{D_5})\frac{\langle\ L \mid w\ ;\ M_+\ \rangle \overset{\delta}{\Longrightarrow} \langle\ L \mid ww'\ ;\ M'_*\ \rangle \langle\ L \mid w\ ;\ N_+\ \rangle \overset{\delta}{\Longrightarrow} \langle\ L \mid ww''\ ;\ N'_*\ \rangle}{\langle\ L \mid w\ ;\ M_+, N_+\ \rangle \overset{\delta}{\Longrightarrow} \langle\ L \mid ww'w''\ ;\ M'_*, N'_*\ \rangle}$$

Remark 3. M is δ-irreducible iff there does not exist M' such that $M \overset{\delta}{\Longrightarrow} M'$.

Proposition 3. *Let Π be a membrane system. If $C \in \mathcal{C}^{\#}(\Pi)$ is tar-irreducible and $C \overset{\delta}{\Longrightarrow} C'$, then C' is δ-irreducible.*

It is worth noting that $C \in \mathcal{C}(\Pi)$ iff C is tar-irreducible and δ-irreducible. According to the standard description in membrane computing, a *transition step* between two configurations $C, C' \in \mathcal{C}(\Pi)$ is given by: $C \Rightarrow C'$ iff C and C' are related by one of the following relations:

either $C \overset{mpr}{\Longrightarrow}; \overset{tar}{\Longrightarrow} C'$,
or $C \overset{mpr}{\Longrightarrow}; \overset{\delta}{\Longrightarrow} C'$,
or $C \overset{mpr}{\Longrightarrow}; \overset{tar}{\Longrightarrow}; \overset{\delta}{\Longrightarrow} C'$.

The three alternatives in defining $C \Rightarrow C'$ are given by the existence of messages and dissolving symbols along the system evolution. Starting from a configuration without messages and dissolving symbols, we apply the "mpr" rules and get an intermediate configuration which is mpr-irreducible; if we have messages, then we apply the "tar" rules and get an intermediate configuration which is tar-irreducible; if we have dissolving symbols, then we apply the dissolving rules and get a configuration which is δ-irreducible. If the last configuration has no messages or dissolving symbols, then we say that the transition relation $\Rightarrow$ is well-defined as an evolution step between the first and last configurations.

Proposition 4. *The relation $\Rightarrow$ is well-defined over the entire set $\mathcal{C}(\Pi)$ of configurations.*

Examples of inference trees, as well as the proofs of the results are presented in [1] and [2].

3 Synchronization Issues in Implementing P Systems

It is evident from the operational semantics that there are several synchronization aspects related to the evolution of a membrane system.

The relationship between the synchronous and the asynchronous in computing systems, particularly in massively-parallel and multiprocessor computing systems, will remain a challenging topic for many years to come. There are reasons to think that the asynchronous approach has some advantages; however the synchronous methodology prevails in the modern computing systems architecture. As if this is not enough, different fields treat the concepts of synchrony and asynchrony

somewhat differently. The main terms (parallelism, concurrency, time) should be clarified in order to discuss the synchronous and asynchronous issues. In our approach we work with a "causal" time (defined as the partial order on some events resulting from their cause-effect relationships) rather a physical time (defined as an independent physical variable related to a clock). The concept of causal time was formulated initially by Aristotle (If nothing happens, no time); it can be useful in systems dealing with events defining cause-effect relationships. The abstract model of a finite state machine corresponds to the model of an asynchronous system evolving in logical time; a possible conversion to a synchronous approach is given by a barrier synchronization (as an engineering solution) in order to manage unpredictable variations of the delays introduced by real physical components. An algorithm (its program) consists of a sequence of steps which perform some actions. Asynchrony is usually treated as the dependence of the number of steps required to obtain the result on the input data. In the case of a fully sequential algorithm (program), such treatment of asynchrony is important only for performance evaluation. Parallel algorithms and programs present new and challenging tasks. Certain steps of an algorithm can be performed concurrently. Representing an algorithm (program) in the form suitable for concurrent implementation is reduced to the cause-effect relationships between the operations (processes, commands) in the algorithm. Thus a parallel specification is a procedure for introducing logical time into the algorithm. An implementation of a global synchronous system can be given by delivering a termination signal from the processors (processes) of the system. Difficulties appear when several processes have a shared resource, and non-synchronized events may occur. A possible solution of a synchronous implementation that eliminates the problems of physical asynchrony is as follows:

- every process can be in two phases: active and passive;
- a process can run only when active;
- to transit from passive to active a process has to receive a signal;
- after an active process executes, it signals other passive processes;

Initially we activate some processes, which after their executions signal passive processes. This repeats until all processes have terminated. Following this scenario, deadlock can occur if the process dependency graph contains cycles. In this scenario, process can be synchronized using a barrier. A *process barrier* is an concurrent abstraction through which multiple processes can be synchronized. Thus a passive process can be considered a process that is waiting at the barrier, and by passing the barrier it becomes an active one.

We can apply this type of synchronization to membrane systems, by allowing a membrane to evolve only after it has passed the barrier. To model this, we use a set of antecedents and a set of descendants for each membrane when describing the system. To apply its rules, a membrane needs to receive signals from all of its antecedents. After it applies its rules, the membrane signals all of its descendants. The set of antecedents specifies how many times a signal needs to be received from each membrane. The set of descendants specifies the membranes that need to be signaled after the application of rules.

Using this mechanism, we can control the relative evolution speed of the antecedents of a membrane. This approach allows to specify that a certain membrane can repeat its step several times before sending its signal to the descendents. In this way we can have a *parameterized synchronization* between membranes, and this aspect could be very useful in modeling biological phenomena. The evolution of a membrane can be described by the following steps which are repeated until no rule can be applied.

1. collect signals from all the antecedents;
2. apply the rules after receiving all the signals;
3. signal all descendants.

4 A Grid-Based Implementation of P Systems

We present here a grid implementation of membrane systems in which we emphasize the notion of computation and synchronization. We employ a synchronization mechanism based on certain preconditions expressing the consistency of the global state of the system. This synchronization mechanism has been introduced to control the dependency relation between membranes. We propose a synchronous model of execution used to coordinate membrane evolution.

To achieve scalability we make use of the grid paradigm *MapReduce*. The paradigm is defined by two main steps: *map*, and *reduce*. The map step allows splitting a task into multiple jobs that execute in parallel on the grid nodes. The reduce step aggregates the result of each job, and returns the task result.

Thus the simulation of a membrane system can be viewed as a grid task. The jobs associated with this task define the execution of each membrane. Hence the number of jobs is equal to the number of membranes. To model the proposed synchronization mechanism between membranes, a communication between jobs is required.

We have selected GridGain [9] as our grid platform because it provides all the required features, and it is easily deployed on multiple platforms. GridGain is a Java-based open source grid computing infrastructure, released under LGPL license. It provides a zero deployment model, meaning that a node can be deployed by running a script, or by creating a node instance. A valuable feature of the system is its support for advanced load balancing and scheduling by providing early and late load balancing that are defined by load balancing and collision (scheduling) resolution. Another important feature is pluggable fault-tolerance with several popular implementations available out-of-the-box. It allows the failover of logic and not only the data. The most notable features of GridGain that we use are: tasks and jobs modeled according to the MapReduce paradigm, communication between grid jobs, and on-demand class loading.

The main steps of the simulation are: (1) Build a membrane system from an specification file; (2) Using the generated membrane system, construct and execute a grid job: (i) *Map*: create a job for each membrane; (ii) *Reduce*: gather all the

```
public class Membrane {
  private List<MembraneLabel> childrenLabels;
  private List<Rule> rules;
  private HashMultiset contents;
  private HashMultiset incomingObjects;
  private MembraneLabel label;
  private MembraneLabel parentLabel;
  private HashMap<MembraneLabel, Integer> antecedents;
  private List<MembraneLabel> descendants;
  private int appliedRules;//number of applied rules in this
      step

  public Membrane()
  //test if the membrane contains a multiset
  public boolean contains(HashMultiset multiset)
  //store a  multiset that resulted in this evolution step
  public void enqueMultiset(HashMultiset multiset)
  //add the objects that resulted in this evolution step
  public void endEvolution()
  //return the list of applicable rules
  public List<Rule> getApplicableRules()
}
```

Fig. 1. Membrane Class

responses from the jobs and create the resulting membrane system.
The simulation repeats step 2 as long as a rule is applied. Each generated job contains an object that describes a membrane from the system. The job is responsible for the correct simulation of the evolution of the membrane. Thus it needs to synchronize with other membranes, and also to apply different rules. The result of the job consists of the final state of the simulated membrane. We have used a modular design for the entities of the system in which we separated the objects defining the *grid behavior* from those defining the *membrane systems*. Thus we implement several abstractions that model various notions such as: *membranes*, *rules*, membrane *objects*, etc. For the grid behavior we define the following concepts: *task*, *job*, *barrier*.

In Figure 1 we describe the members and main methods of class Membrane. The object is responsible only for operations that modify the contents of a membrane. The evolution logic is implemented using the Rule and EvolutionVisitor objects. To model the rules of a membrane system we used a extensible approach. Each rule can be seen as a list of constraints; a constraint is responsible for checking if its precondition is valid (via method *check*), and for applying its postcondition on a membrane (via method *apply*).

```
public class Rule extends RuleConstraint {
  List<RuleConstraint> constraints;

  public Rule()
  public void apply(Membrane membrane) {
    for (RuleConstraint constraint : constraints) {
      constraint.apply(membrane);
    }
  }
  public boolean check(Membrane membrane) {
    boolean isApplicable = true;
    Iterator<RuleConstraint> iter = constraints.iterator();
    while (isApplicable && iter.hasNext()) {
      isApplicable = isApplicable && iter.next().check(membrane)
          ;
    }
    return isApplicable;
  }
}
```

Fig. 2. Rule Class

From a software engineering perspective, the rule follows the composite pattern. In Appendix[1], we present the RuleConstraint class and in how a constraint multiset can be implemented. The main methods of the Rule class are presented in Figure 2. Using these abstractions we can easily implement rules with various ingredients, only by describing constraints and aggregating them into a new type of Rule. The evolution of a membrane is performed by the EvolutionVisitor object described. The method *localMembraneEvolution* defines the logic of a single step of evolution. A step is simulated by the repeated application of rules.

A grid task is defined by the class PsTask, which follows the MapReduce paradigm. The method *split* takes as input a membrane system, and for each membrane creates a job that will be executed on the grid. The method *reduce* receives a list of job results that contain membranes, and assembles them in a membrane system.

A grid job is described by the PsJob object (in Appendix). This object contains a membrane which holds the data, and a barrier used for synchronization. The main method of this class is *execute*, in which the evolution of a membrane is executed. The evolution consists of a three-step loop: (i) wait at the barrier for incoming signals, (ii) after receiving the signals, apply the rules, and (iii) after applying the rules, signal the descendants. The result of the job is a maximally parallel step of the membrane.

[1] Available online at `www.gcn.us.es/12bwmc_proceedings`

```
public class PsTask{
  public MembraneSystem reduce(List results){
    MembraneSystem result = new MembraneSystem();
    int appliedRules = 0;
    for (GridJobResult gridJobResult : results) {
      Membrane data = gridJobResult.getData();
      result.getMembranes().put(data.getLabel(), data);
      appliedRules += data.getAppliedRules();
    }
    result.setAppliedRules(appliedRules);
    return result;
  }
  protected List<GridJob> split(MembraneSystem arg){
    List<PsJob> jobs = new ArrayList<PsJob>();
    for(Membrane mbr : arg.getMembranes().values()){
      jobs.add(new PsJob(mbr));
    }
    return jobs;
  }
}
```

Fig. 3. PsTask Class

Membrane synchronization is achieved by using a special form of barrier. The barrier waits to be signaled from each antecedent membrane a specified number of times. After this, it releases the job that called the method *waitAt*. The barrier also listens for termination signals. When it receives such a signal it informs the waiting job that it should finish its execution.

5 Example

We provide a simple example to illustrate the developed simulator. The system is composed of two membranes $m1$, $m2$. Membrane $m1$ contains a^{2000} and has rules $a \to b$, and $b^2 \to d$, while membrane $m2$ contains $a^{40000}b^{1000}c^{5000}$ and has rules $a^2 \to b$, and $c^2 \to d$. The signaling part is denoted by the contents of *wait*, and *signal*. Those include a sequence of membranes and the number of times they have to signal. Notice that $m2$ has to wait to be signaled by $m1$ two times before it can apply a rule. The parent of $m2$ is $m1$, which is the skin membrane.

```
/* PsGrid input file */
membrane m1 /*name of the membrane*/ :
      skin /*name of the parent*/{
  children {
    m1  /*name of children*/
  }
```

```
  contents {
    a^{2000}/*contents of the membrane*/
  }
  rules {
    /*rules of the membrane*/
    [a^{1} ==> b^{1}]
    [b^{2} ==> d^{1}]
  }
  wait{
    /*the antecedents*/
  }
  signal{
    m2 /*the descendants*/
  }
}
membrane m2 : m1{
  children {
  }
  contents {
    a^{40000}b^{1000}c^{5000}
  }
  rules {
    [a^{2} ==> b^{1}]
    [c^{2} ==> d^{1}]
  }
  wait{
    m1^{2}
  }
  signal{
  }
}
```

We also present the log from each node of the grid. The log shows the order in which membrane jobs arrive at each node, and the actions they execute. The number of rule applications executed in a certain step is written at the end of the lines (after #). Notice that the job ends if it receives a terminate signal, or if the membrane did not apply any rules in this step.

```
[20:26:56,843][INFO ][gridgain-#6%null%][PsJob] Received membrane with
    contents :m1:[a^{2000} ] [[Rule:  in m1[a  -> b ], Rule:  in m1[b^{2}
     -> d ]]] #0
[20:26:56,843][INFO ][gridgain-#10%null%][PsJob] Received membrane with
    contents :m2:[b^{1000} c^{5000} a^{40000} ] [[Rule:  in m2[a^{2}  -> b
     ], Rule:  in m2[c^{2}  -> d ]]] #0
[20:26:56,843][INFO ][gridgain-#10%null%][PsJob] Waiting at barrier:m2
[20:26:56,843][INFO ][gridgain-#6%null%][PsJob] Waiting at barrier:m1
[20:26:56,843][INFO ][gridgain-#6%null%][PsJob] Passing the barrier:m1
[20:26:56,875][INFO ][gridgain-#6%null%][PsJob] Sending signal to
    descendants
[20:26:56,890][INFO ][gridgain-#6%null%][PsJob] After evolution :m1:[b
    ^{2000} ] [[Rule:  in m1[a  -> b ], Rule:  in m1[b^{2}  -> d ]]] #2000
[20:26:56,890][INFO ][gridgain-#6%null%][PsJob] Waiting at barrier:m1
[20:26:56,890][INFO ][gridgain-#6%null%][PsJob] Passing the barrier:m1
[20:26:56,890][INFO ][gridgain-#10%null%][PsJob] Passing the barrier:m2
[20:26:56,906][INFO ][gridgain-#6%null%][PsJob] Sending signal to
    descendants
[20:26:56,921][INFO ][gridgain-#6%null%][PsJob] After evolution :m1:[d
    ^{1000} ] [[Rule:  in m1[a  -> b ], Rule:  in m1[b^{2}  -> d ]]] #1000
[20:26:56,921][INFO ][gridgain-#6%null%][PsJob] Waiting at barrier:m1
[20:26:56,921][INFO ][gridgain-#6%null%][PsJob] Passing the barrier:m1
[20:26:56,921][INFO ][gridgain-#6%null%][PsJob] Sending signal to
    descendants
```

```
[20:26:56,921][INFO ][gridgain-#6%null%][PsJob] After evolution :m1:[d
    ^{1000} ] [[Rule:  in m1[a  -> b ], Rule:  in m1[b^{2}  -> d ]]] #0
[20:26:57,062][INFO ][gridgain-#10%null%][PsJob] Sending signal to
    descendants
[20:26:57,062][INFO ][gridgain-#10%null%][PsJob] After evolution :m2:[d
    ^{2500} b^{21000} ] [[Rule:  in m2[a^{2}  -> b ], Rule:  in m2[c^{2}
    -> d ]]] #22500
[20:26:57,062][INFO ][gridgain-#10%null%][PsJob] Waiting at barrier:m2
[20:26:57,062][INFO ][gridgain-#10%null%][PsJob] Passing the barrier:m2
[20:26:57,062][INFO ][gridgain-#10%null%][PsJob] Sending signal to
    descendants
[20:26:57,062][INFO ][gridgain-#10%null%][PsJob] After evolution :m2:[d
    ^{2500} b^{21000} ] [[Rule:  in m2[a^{2}  -> b ], Rule:  in m2[c^{2}
    -> d ]]] #0
```

Fig. 4. PsGrid Screen After Executing the Example

The simulator has a simple but flexible graphical interface. A screen-shot after executing a simulation is presented in Figure 4. The first row presents the initial configuration of the membrane system. The second row presents the contents of the membranes after the simulation.

Even though this example is simple, the implementation can benefit from several features of GridGain, and provide a complex parallel implementation of membrane systems. The main points are that the implementation is faithful to the

formal description of the membrane systems, and it is also scalable to a high number of membranes (which is the case in cell biology simulations).

6 Conclusion

Hierarchies are often used in modeling and simulation for computational biology. A hierarchical perspective of the cell considers components structured into classes of similar kinds, e.g. golgi, ER, and nucleus form organelles, i.e. membrane-bound compartments of the cell. New models of membrane systems need to be simulated on complex hardware systems in order to provide a valuable feedback to biologists. Membrane computing is a branch of natural computing using an explicit hierarchical description coming exactly from the structure and functioning of the living cell. The main areas where membrane computing has been used as a modeling framework (biology and bio-medicine, linguistics, economics, computer science, etc.) are presented in [5]. In that volume, several implementations (mainly using sequential computational environments) for simulating various types of cell-like membrane systems are presented in [6]. We consider that the simulation of P systems with sequential computers is a complex task because membrane systems are intrinsically parallel and nondeterministic computational devices, and their computation trees are difficult to store and handle with one processor. Therefore it is necessary to look for parallel and scalable implementations able to simulate as close as possible the formal description of the membrane systems.

In this paper we present a faithful parallel implementation of membrane systems using GridGain, emphasizing on the synchronization problems appearing in membrane computing. Thus we hope to offer a more suitable simulator for membrane systems, opening a new possibility of using membrane computing as a parallel and nondeterministic modeling framework for addressing structural and dynamical aspects of complex systems modeling phenomena in cell biology where huge number of elements are used (some phenomena are presented in [5]).

In the papers devoted to membrane systems it is not mentioned how the membranes (or groups of membranes) interact or synchronize. The usual thinking is that membrane systems are synchronized locally (a step of a membrane is given by the parallel application of rules), and behave asynchronously at the global level. We emphasize here the global aspects, by adding a form of parameterized barrier synchronization between membranes.

There are several software simulators for P systems; however almost all of them are on sequential hardware, and so they do not match the parallel nature of P systems. A parallel implementation of P systems (one of the very few, if not the unique working in a parallel hardware setting) is presented in [4]. It uses a cluster of 64 dual processors, and an MPI library in order to describe the communication and synchronization of parallel processes. In that parallel simulator, the rules are implemented as threads. At the system initialization phase, one thread is created for each rule. Within one membrane, several rules can be applied concurrently.

This parallelism between rule applications within one membrane is modeled with multithreading. Rule applications are performed in terms of rounds. To synchronize each thread (rule) within the system, two barriers implemented as mutexes are associated with a thread. At the beginning of each round, the barrier that the rule thread is waiting on is released by the primary controlling thread. After the rule application is done, the thread waits for the second barrier, and the primary thread locks the first barrier. During the following round it would repeat the above procedure, releasing and locking alternating barriers. Since many rules are executing concurrently and they are sharing resources, a mutual exclusion algorithm is necessary.

The communication and synchronization between membranes is implemented using the Message Passing Interface library of functions for parallel computation. The execution is performed in terms of rounds. At the end of each round, every membrane exchanges messages with all its children and parent before proceeding to the next round. Another concern is the termination detection problem. When the system is no longer active, there is no rule in any membrane that is applicable, all the membranes must be able to be informed, and to terminate. Once the skin membrane detects the termination, it broadcasts this information to all the other membranes. Thereafter, the system terminates and the output is written to a specified file. Fundamental distributed algorithms in the framework of membrane systems are presented in [3].

References

1. Andrei, O., Ciobanu, G. and Lucanu, D. Operational Semantics and Rewriting Logic in Membrane Computing, *Electronic Notes of Theoretical Computer Science* vol.156, 57–78, 2006.
2. Andrei, O., Ciobanu, G. and Lucanu, D. A Rewriting Logic Framework for Operational Semantics of Membrane Systems. *Theoretical Computer Science* vol.373, 163–181, 2007.
3. G. Ciobanu. Distributed Algorithms over Communicating Membrane Systems. *Biosystems* vol.70, Elsevier, 123–133, 2003.
4. G. Ciobanu, W. Guo. P Systems Running on a Cluster of Computers. *Membrane Computing*, Lecture Notes in Computer Science vol.2933, Springer, 123–139, 2004.
5. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.). *Applications of Membrane Computing*. Natural Computing Series, Springer, 2006.
6. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Available Membrane Computing Software. In *Applications of Membrane Computing* [5], Springer, 411–436, 2006.
7. Păun, Gh. *Membrane Computing. An Introduction*. Springer, 2002.
8. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
9. Website GridGain: `http://gridgain.com`.

Self-constructing Recognizer P Systems

Daniel Díaz-Pernil[1], Francisco Peña-Cantillana[2],
Miguel A. Gutiérrez-Naranjo[2]

[1]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics - University of Sevilla, 41012, Spain
sbdani@us.es

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
frapencan@gmail.com, magutier@us.es

Summary. Usually, the changes produced in the membrane structure of a P system are considered side effects. The output of the computation is encoded as a multiset placed in a specific region and the membrane structure in the halting configuration is not considered important. In this paper we explore P systems where the target of the computation is the construction of a new membrane structure according its set of rules. The new membrane structure can be considered as the initial one of a new *self-constructed* P system. We focus on the self-construction of recognizer P systems and illustrates the definition with a study of the self-construction P systems working as decision trees for solving Machine Learning decision problems.

1 Introduction

In many Membrane Computing models, changing the membrane structure of a P system along the computation is a common process. The changes are produced via division of membranes (based on *cellular mitosis*), via creation of new membranes from objects (based on *cellular autopoiesis*, see [7]) or dissolution of membranes. The rules producing such changes have been deeply studied and the capability of the P systems for solving hard problems are linked to the use of such rules (see, e.g., [4, 5, 16]).

Nonetheless, the changes of the membrane structure produced along a computation are not considered a target itself. The changes are usually produced in order to compute an *output*, which is usually encoded as a multiset (or as a single distinguished object) in the corresponding output region. The membrane structure obtained in the halting configuration is not important. It is merely a collateral effect.

In this paper, we focus on P systems where the target of the computation is exactly the opposite to the usual one. We study P systems whose aim is to

develop a membrane structure. Such P systems will take a multiset as *input* and they will change their membrane structure according to such input, the set of rules and the non-deterministic choices, if any. The membrane structure obtained in the halting configuration will be considered as the *output* of the computation. This new membrane structure, together the remaining ingredients of the P system (alphabet, set of labels, set of rules, ...) can be considered as a new P system, able to receive a new *input* and perform a new computation. In this way, we will consider that this *second* P system (which is similar to the original one, but with a new initial membrane structure) has been self-constructed, since a (potentially) complex membrane structure has been obtained from a simple one (maybe from an initial membrane structure with the skin as unique membrane) according to the application of their own rules. Of course, different final membrane structures may be obtained from different inputs, but also with the same input due to the non-determinism.

The self-construction of a complex membrane structure can be a target by itself, as shown in [3, 6], but in this paper, the self-constructed P system is thought for a second use. From this general target, we focus here on the self-construction of *recognizer P systems*, i.e., the P system with this new membrane structure can be now used as recognizer P systems for solving decision problems in the usual way: An instance of the decision problem is provided to the P system as an input encoded as an appropriate multiset and an object *yes* or *no* (but no both) is sent to the environment in the last step of the computation.

In this way, two different uses for the P system are considered:

- Firstly, given a P system, a multiset is placed in the corresponding input membrane and the computation starts. According to the input and the non deterministic choice of applicable rules, the initial membrane structure is modified along the computation. As usual, a halting configuration is reached if no more rules can be applied. The self-construction of the recognizer P system is finished.
- Secondly, we consider a new computation of the P system, but in this stage, the membrane structure obtained in the halting configuration of the previous stage is considered as the initial one. This new computation also needs a new input, which is placed in the corresponding input membrane. In this stage, the *output* will be a specific object (*yes* or *no*, but no both) which is placed in the output region in the halting configuration.

The paper is organized as follows: Next, we recall the definition of recognizer P system used in this paper. In Section 3, our case study is presented, the self-construction of a P system from a training set which works as a decision tree and the classification (decision problem) of new instances as in Machine Learning theory. We provide the formal framework, the general construction of the P systems, an example and some theoretical considerations. Finally, Section 4 finishes the paper with some conclusions.

2 Recognizer P Systems

Recognizer P systems were introduced in [11] and they are the natural framework to study and solve decision problems, i.e., problems were a Boolean total function θ_X must be defined on a set of instances I_X. Recognizer P systems are associated in a natural way with P systems with *input* and with external *output*, i.e., each instance of the problem is codified by a multiset placed in an *input membrane*. The output of the computation (*yes* or *no*) is sent to the environment. Due to the non-determinism, the definition of recognizer P system claims that the output of *all* the computations must be the same. Since one can find slightly different approaches in the literature (see [9, 15]), we recall the definition used in this paper:

Definition 1. *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where: (a)* Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \ldots, p$*, and initial multisets* $w_1, \ldots, w_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ*; the initial multisets are over* $\Gamma - \Sigma$*; and (c)* i_Π *is the label of a distinguished (input) membrane.*

Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, w_1, \ldots, w_{i_\Pi} \cup m, \ldots w_p)$.

Definition 2. *A* recognizer P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output such that:*

1. *The working alphabet contains two distinguished elements yes, no.*
2. *All its computations halt.*
3. *If* $\mathcal{C}$ *is a computation of* Π*, then either some object yes or some object no (but not both) must have been released into the environment, and only in the last step of the computation. We say that* $\mathcal{C}$ *is an accepting computation (respectively, rejecting computation) if the object yes (respectively, no) appears in the external environment associated to the corresponding halting configuration of* $\mathcal{C}$*.*

3 A Case Study: Decision Trees

Decision trees is one of the most widely used structures in Computer Science. They are used to classify an input by sorting it down the tree from the root to some leaf node, which provides the classification of the instance. Instances are usually written as sets of pairs $\langle Attribute, Value \rangle$ and each node in the tree determines a test of some attribute. Each branch descending from that node corresponds to one of the possible value for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified in this node and moving down the branch corresponding to the value of the instance in this attribute. The leaves are labelled with values of the classification and they are the output associated to the instances that reach them. In this way, if the possible classifications are

Table 1. A classic example of training set adapted from [14].

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

YES and NO, a decision tree can be thought as a Boolean mapping on the set of instances which *decides* the classification of the instance according to the values of the attributes.

Let us illustrate the process with a classic example adapted from [14]. It consists on a database with fourteen days. Each day is represented by the values of the attributes *Outlook*, *Temperature*, *Humidity* and *Wind*. Each day has also associated its classification with respect to the attribute *PlayTennis* (see Table 1). From a learning point of view, the database can be seen as a training set. The target is to generate a decision tree from this database which can be used to classify *new* instances.

Figure 1 shows a decision tree consistent with the training set shown in Table 1, i.e., a tree which classifies correctly all the examples in the training set. According to this tree, a day with $Outlook = Sunny$, $Temperature = Cool$, $Humidity = Normal$ and $Wind = Strong$ (which does not belong to the training set) will be classified as YES.

3.1 The P System Model

The definition of self-construction in P system is independent of the P system model, i.e., it can be considered in the framework of cell-like, tissue-like or whatever other graph structure and it can be adapted to different semantics. The unique restriction that the P system must satisfy is the ability of modifying the initial membrane structure according to the input. In this way, the concept can be considered in many scenarios.

In this paper, we will illustrate the definition with a P system model were the data are encoded as strings [1, 13] and the changes in the membrane structure are performed via membrane creation [5, 10].

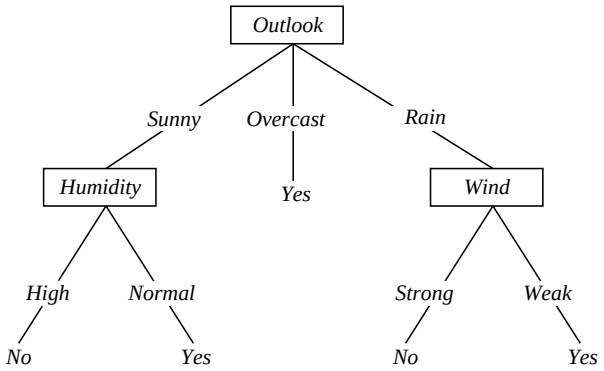


Fig. 1. An example of tree obtained from the training set shown in 1. The image is available from `http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html`.

Formally, a *self-constructing P system with strings and membrane creation* is a construct of the form $\Pi = (O, H, L, R)$ where:

1. O is the alphabet of *objects*;
2. H is a finite set of labels;
3. L is a finite languages over O;
4. R is a finite set of *rules*, of the following forms:
   a) $[w_p + w_q \rightarrow \sum_k w_k]_h$ where $h \in H$, w_p, w_q and w_k are strings over O. These are *2-cooperative evolution rules*: The simultaneous occurrence of the strings w_p and w_q in the membrane h produces a finite set of strings in the same membrane. As usual, w_p and w_q are consumed.
   b) $w_p[\,]_h \rightarrow [w_q]_h$ where $h \in H$; w_p and w_q are strings over O. These are *send-in communication rules*. A string is introduced in the membrane possibly modified.
   c) $[w_p]_h \rightarrow [\,]_h\, w_q$ where $h \in H$; w_p and w_q are strings over O. These are *send-out communication rules*. A string is sent out of the membrane possibly modified.
   d) $[a \rightarrow [M]_{h_2}]_{h_1}$ where $h_1, h_2 \in H$, $a \in O$ and M is a finite language over O. These are *creation rules*. An object a placed in a membrane with label h_1 creates a new membrane with label h_2. This new membrane has associated an initial finite language M.

We will consider that the *self-constructing P system with strings and membrane creation* always has a unique membrane (the skin) in the initial membrane structure, such membrane is the input membrane and the output region is the environment. The multiset L is placed in the skin at the initial configuration. Rules are applied according to the following principles:

- Rules are used as usual in the framework of Membrane Computing, that is, in a maximal parallel way. In one step, each string in a membrane can only be used

for one rule (non deterministically chosen when there are several possibilities), but any string which can evolve by a rule of any form must do it (with the restrictions below indicated).
- All the strings which are not involved in any of the operations to be applied remain unchanged.
- The rules associated with the label h are used for all membranes with this label, irrespective of whether or not the membrane is an initial one or it was obtained by creation.
- Several rules can be applied to different objects in the same membrane simultaneously.

3.2 Self-constructing P Systems for Decision Trees Learning

Next, we will provide a family of *self-constructing P system with strings and membrane creation* for Decision Trees Learning. Such self-constructed P system will receive instances of the problem and will output *yes* or *no*, i.e., they are tools for solving decision problems.

In a first stage, the P system takes a finite language, codifying a *training set*, as input. Each example in the training set will be encoded as a string. The set of these strings will be the initial language L and it will be placed in the unique initial membrane of the P system, as defined above. After a finite number of steps, the computation halts and the membrane structure has been (probably) modified. The P system with this halting membrane structure is the recognizer P system which has been self-constructed according with the training set provided as input. The self-constructed recognizer P system with the membrane structure obtained in the halting configuration is now prepared to receive an *instance* of the decision problem and will provide an answer *yes* or *not*.

Let us start by considering a training set $D = \{(v_1, c_1), \ldots, (v_n, c_n)\}$ where, for each $i \in \{1, \ldots, n\}$, v_i is a tuple of pairs $\langle Attribute, Value\rangle$ and $c_i = \{YES, NO\}$ is the classification for a concept[1]. We will consider a set of attributes $ATR = \{A_1, \ldots, A_k\}$ and, for each $i \in \{1, \ldots, k\}$, $VAL_i = \{v_i^1, \ldots, v_i^{j_i}\}$ is the set of values of the attribute A_i. We will consider that the sets VAL_i are disjoint pairwise. We will also consider $VAL = VAL_1 \cup \ldots \cup VAL_k$ and $\Gamma = ATR \cup VAL \cup \{YES, NO\}$. The set Γ will be called a *training set alphabet*. Notice that many different training sets can have the same alphabet Γ.

By using this notation, we will represent each example (v_i, c_i) as a string over the set Γ and a training set can be considered as a finite language over this set. The example (v_i, c_i) will be represented by the string of $2k + 1$ symbols $A_1 v_1^i A_2 v_2^i \ldots A_k v_k^i c_i$, where v_j^i is the value of the attribute A_j in the $i - th$ example of the training set and $c_i \in \{YES, NO\}$ is the value of the classification. For example, the string

[1] A formal description of the principles of Machine Learning is out of the scope of this paper. A detailed introduction can be found in, e.g., [8].

$$Outlook\, Sunny\, Temperature\, Hot\, Humidity\, High\, Wind\, Weak\, NO$$

is the representation of the first example of Table 1.

In the first stage, the finite language codifying the training set is placed in the skin and the self-construction starts. When it finishes, the halting membrane structure is prepared for accepting new inputs and *deciding* on it. The new input will be similar to the encoding of an example, but the last object of the string will be ? instead of YES or NO. For example, in order to know the classification of a day not with $Outlook = Sunny$, $Temperature = Cool$, $Humidity = Normal$ and $Wind = Strong$, the string

$$Outlook\, Sunny\, Temperature\, Cool\, Humidity\, Normal\, Wind\, Strong\, ?$$

will be placed in the input membrane (the skin) and a new computation will start.

Next we provide the formal definition of a *self-constructing P system with strings and membrane creation* associated to a training set Γ. The P system is a 4-uple $\Pi = (O, H, L, R)$ where:

1. $O = \Gamma \cup \{YES_{aux}, YES_{act}, ?, NO_{aux}, NO_{act}, new\}$ is the alphabet of objects;
2. $H = \{skin\} \cup VAL$. The possible labels are the values of the attributes plus the initial label $skin$;
3. $L = \{YES_{aux}, NO_{aux}\}$ Two strings, each of them with only one object, are placed in the skin in the initial configuration;
4. We split the set R of rules into two groups, the rules used in the self-construction stage and the rules used in the decision stage:

Rules for the self-construction stage.

R1. $\left.\begin{array}{l}[\overline{x}YES + YES_{aux} \to \overline{x}YES + YES_{act}\,]_h \\ [\overline{x}NO + NO_{aux} \to \overline{x}NO + NO_{act}\,]_h\end{array}\right\}$ for $h \in H$.

where $\overline{x}$ is a string over Γ composed by pairs $(Attribute, Value)$. If the string $\overline{x}YES$ and YES_{aux} occur simultaneously in the same membrane, then $\overline{x}YES$ remains unchanged, but YES_{aux} is consumed and YES_{act} is produced. Analogously for the NO case.

R2. $[YES_{act} + NO_{act} \to new]_h$ for $h \in H$.

If the strings YES_{act} and NO_{act} are placed simultaneously in the same membrane, both are consumed and the string new is produced.

R3. $[new + \overline{x}A_i\overline{y} \to \overline{x}A_i\overline{y} + v_1 + \ldots + v_s]_h$ for $h \in H$.

If the strings new and $\overline{x}A_i\overline{y}$ occur in the same membrane (where $\overline{x}A_i\overline{y}$ is a string including the object A_i, which denotes an attribute), then the string new is consumed, the string $\overline{x}A_i\overline{y}$ remains unchanged and all the strings v_j^i from VAL_i are produced. Let us notice that this set of rules produces a high

degree of non-determinism. On the one hand, many different strings $\overline{x}A_i\overline{y}$ can simultaneously occur in the same membrane and, on the other hand, several A_i may be chosen from the same string. Nonetheless, since there can exists at most only one string *new* in each membrane at each time unit, only one of these rules will be applied.

R4. $[v \to [\, YES_{aux}\, NO_{aux}]_v\,]_h$ for $h \in H$
Each string $v \in VAL$ creates a new membrane. Such membrane will have v as a label and it will contain the strings YES_{aux} and NO_{aux}.

R5. $\overline{x}A_iv\overline{y}\,[\,]_v \to [\,\overline{xy}\,]_v$ for $A_i \in ATR$ and $v \in VAL_i$
Each string $\overline{x}A_iv\overline{y}$ (where A_i is an object which denotes an attribute and v is the next symbol in the string, denoting one of the values of A_i) out of a membrane with label v will be sent into the membrane. The application of the rule will transform the string into $\overline{xy}$, which is $\overline{x}A_iv\overline{y}$ after deleting the substring A_iv. These rules are applied in parallel and several strings can cross out the same membrane simultaneously.

Rules for the decision stage.

R6. $\overline{x}Av\overline{y}?\,[\,]_v \to [\,\overline{xy}?\,]_v$ for $v \in VAL$.
If a string ended with ? and containing an object $v \in VAL$ is out of a membrane with label v, then the string is sent into the membrane. The application of the rule also produces a change in the string since the object v and the previous object in the string (the object A denoting the corresponding attribute) are deleted.

R7. $\left.\begin{array}{l}[\overline{x}? + YES_{act} \to YES_{out} + YES_{act}\,]_h \\ [\overline{x}? + NO_{act} \to NO_{out} + NO_{act}\,]_h\end{array}\right\}$ for $h \in H$.
where $\overline{x}$ is a string over Γ composed by pairs $(Attribute, Value)$. If the string $\overline{x}?$ and YES_{act} occur simultaneously in the same membrane, then YES_{act} remains unchanged, but $\overline{x}?$ is consumed and YES_{out} is produced. Analogously for the NO case.

R8. $\left.\begin{array}{l}[\,YES_{out}\,]_h \to [\,]_h\, YES_{out} \\ [\,NO_{out}\,]_h \to [\,]_h\, NO_{out}\end{array}\right\}$ for $h \in H$.
when an object YES_{out} (resp. NO_{out}) is produced, the decision is made. This set of rules sends such object from the membrane where is produced to the environment. Such objects are the answers to the decision problem

$Outlook\, sunny\, Temperature\, hot\, Humidity\, high\, Wind\, weak\, NO$
$Outlook\, sunny\, Temperature\, hot\, Humidity\, high\, Wind\, strong\, NO$
$Outlook\, overcast\, Temperature\, hot\, Humidity\, high\, Wind\, weak\, YES$
$Outlook\, rain\, Temperature\, mild\, Humidity\, high\, Wind\, weak\, YES$
$Outlook\, rain\, Temperature\, cool\, Humidity\, normal\, Wind\, weak\, YES$
$Outlook\, rain\, Temperature\, cool\, Humidity\, normal\, Wind\, strong\, NO$
$Outlook\, overcast\, Temperature\, cool\, Humidity\, normal\, Wind\, strong\, YES$
$Outlook\, sunny\, Temperature\, mild\, Humidity\, high\, Wind\, weak\, NO$
$Outlook\, sunny\, Temperature\, cool\, Humidity\, normal\, Wind\, weak\, YES$
$Outlook\, rain\, Temperature\, mild\, Humidity\, normal\, Wind\, weak\, YES$
$Outlook\, sunny\, Temperature\, mild\, Humidity\, normal\, Wind\, strong\, YES$
$Outlook\, overcast\, Temperature\, mild\, Humidity\, high\, Wind\, strong\, YES$
$Outlook\, overcast\, Temperature\, hot\, Humidity\, normal\, Wind\, weak\, YES$
$Outlook\, rain\, Temperature\, mild\, Humidity\, high\, Wind\, strong\, NO$

Fig. 2. Finite language encoding the training set from Table 1.

3.3 An example

As an example of self-construction P systems for Decision Tree Learning, let us consider the training set from Table 1. According to the encoding previously described, such training set can be written as shown in Fig. 2.

Let us consider an initial configuration C_0 which has only one membrane with label *skin*. Such membrane contains the language codifying the training set from Fig. 2, together with YES_{aux} and NO_{aux}. From this initial configuration only two rules from **R1** are applied. The application of such rules consumes YES_{aux} and NO_{aux} and produces YES_{act} and NO_{act}. In the second step of the computation, only the rule from **R2** is applied. The objects YES_{act} and NO_{act} are consumed and *new* appears in the skin. In this way, the configuration C_2 has only one membrane, the *skin*, where the codification of the training set and the object *new* are placed.

From this configuration C_2, one and only one of the rules from **R3** is nondeterministically chosen and applied. In the choice, one of the strings encoding an example from the training set is taken and in this string, one of the objects encoding an attribute is also selected. Let us suppose that the string

$$Outlook\, sunny\, Temperature\, hot\, Humidity\, high\, Wind\, weak\, NO$$

is chosen and the object *Outlook* is selected. The application of the rule consumes the object *new*, keeps unchanged the string encoding the example and three new objects *rain*, *sunny* and *overcast* appear. Therefore, the configuration C_3 has only one membrane, the *skin*, where the codification of the training set and the objects *rain*, *sunny* and *overcast* are placed.

In the next step, the changes in the membrane structure start. The objects *rain*, *sunny* and *overcast* create new membranes. Each membrane has the objects YES_{aux} and NO_{aux} inside and the corresponding value *rain*, *sunny* or *overcast* as label, i.e.,

$$C_4 = \left[\begin{array}{cc} TS & [\, YES_{aux}\, NO_{aux}\,]_{rain} \\ [\, YES_{aux}\, NO_{aux}\,]_{sunny} & [\, YES_{aux}\, NO_{aux}\,]_{overcast} \end{array} \right]_{skin}$$

where TS represents the language encoding the training set. In the next step, rules from **R5** are applied. All the strings in the skin are sent into the new membranes with slight changes. These new strings in the elementary membranes are the following. The set TR_{sunny} of strings in the membrane with label *sunny* is

$$\begin{array}{l} Temperature\, hot\, Humidity\, high\, Wind\, weak\, NO \\ Temperature\, hot\, Humidity\, high\, Wind\, strong\, NO \\ Temperature\, mild\, Humidity\, high\, Wind\, weak\, NO \\ Temperature\, cool\, Humidity\, normal\, Wind\, weak\, YES \\ Temperature\, mild\, Humidity\, normal\, Wind\, strong\, YES \end{array}$$

the set TR_{rain} of strings in the membrane with label *rain* is

$$\begin{array}{l} Temperature\, mild\, Humidity\, high\, Wind\, weak\, YES \\ Temperature\, cool\, Humidity\, normal\, Wind\, weak\, YES \\ Temperature\, cool\, Humidity\, normal\, Wind\, strong\, NO \\ Temperature\, mild\, Humidity\, normal\, Wind\, weak\, YES \\ Temperature\, mild\, Humidity\, high\, Wind\, strong\, NO \end{array}$$

and, finally the set $TR_{overcast}$ of strings in the membrane with label *overcast* is

$$\begin{array}{l} Temperature\, hot\, Humidity\, high\, Wind\, weak\, YES \\ Temperature\, cool\, Humidity\, normal\, Wind\, strong\, YES \\ Temperature\, mild\, Humidity\, high\, Wind\, strong\, YES \\ Temperature\, hot\, Humidity\, normal\, Wind\, weak\, YES \end{array}$$

In the configuration C_5, the membrane with label *sunny* has five strings encoding examples and two objects YES_{aux} and NO_{aux}. The situation is similar to the initial configuration, where the membrane *skin* had fourteen strings encoding examples. Let us consider that from the configuration C_8, the chosen rule from **R3** takes the string

$$Temperature\, hot\, Humidity\, high\, Wind\, weak\, NO$$

and the object $Humidity$. In C_9, two new membranes appear inside the membrane with label *sunny*, one of them with membrane *high* and the other one with label *normal*. In the configuration C_{10}, this new membrane with label *high* contains the objects YES_{aux} and NO_{aux} and the set of strings $TR_{sunny+high}$

$$\begin{array}{l} Temperature\, hot\, Wind\, weak\, NO \\ Temperature\, hot\, Wind\, strong\, NO \\ Temperature\, mild\, Wind\, weak\, NO \end{array}$$

analogously, the new membrane with label *normal* contains the objects YES_{aux} and NO_{aux} and the set of strings $TR_{sunny+normal}$

$$Temperature\, cool\, Wind\, weak\, YES$$
$$Temperature\, mild\, Wind\, strong\, YES$$

In a similar process, if the chosen rule from **R3** in the membrane with label *rain* selects an object $Wind$ from the taken string, then, such membrane will have two new membranes inside in the configuration C_{10}. One on them, with label *strong* will contain the objects YES_{aux} and NO_{aux} and the set of strings $TR_{rain+strong}$

$$Temperature\, cool\, Humidity\, normal\, NO$$
$$Temperature\, mild\, Humidity\, high\, NO$$

The second new membrane, with label *weak* will contain the objects YES_{aux} and NO_{aux} and the set of strings $TR_{rain+weak}$

$$Temperature\, mild\, Humidity\, high\, YES$$
$$Temperature\, cool\, Humidity\, normal\, YES$$
$$Temperature\, mild\, Humidity\, normal\, YES$$

Let us consider now the membrane with label *overcast* in the configuration C_5. It contains the objects YES_{aux} and NO_{aux} and the set of strings $TR_{overcast}$. In the next step, the object YES_{aux} is transformed into YES_{act} by application of one rule from **R1**, but NO_{aux} keeps unchanged, since there is no string with NO as the last object. This means that no more rules can be applied and the computation in this membrane finishes. The same reasoning is valid for the remaining membranes where all the strings end in YES or NO. In this way, the configuration C_{11} is a halting one and the decision P systems is already constructed (see Fig. 3).

This self-constructed recognizer P system can be used now for deciding on new instances. Let us consider a new computation. The target is to obtain a Boolean answer as a classification for the day with $Outlook = Sunny$, $Temperature = Cool$, $Humidity = Normal$ and $Wind = Strong$. In such way, the string

$$Outlook\, Sunny\, Temperature\, Cool\, Humidity\, Normal\, Wind\, Strong\, ?$$

will be placed in the input membrane (the skin) and this is the unique string in the skin in the new configuration C_0. The corresponding rule from **R6** is applied and the string is sent to the membrane with label *sunny* slightly modified:

$$Temperature\, Cool\, Humidity\, Normal\, Wind\, Strong\, ?$$

In the next step, a new rule from **R6** is applied and the string $Temperature\, Cool\, Wind\, Strong\, ?$ is placed in the membrane with label *normal* in the configuration C_2. Since YES_{act} occurs in this membrane, the string is consumed and YES_{out} appears in the configuration C_3. By three applications of rules from **R7**, the object YES_{out} is sent to the environment and the configuration C_6 is a halting configuration. The answer YES_{out} is sent to the environment in the last step of the computation and it is the answer corresponding to the input in this recognizer P system.

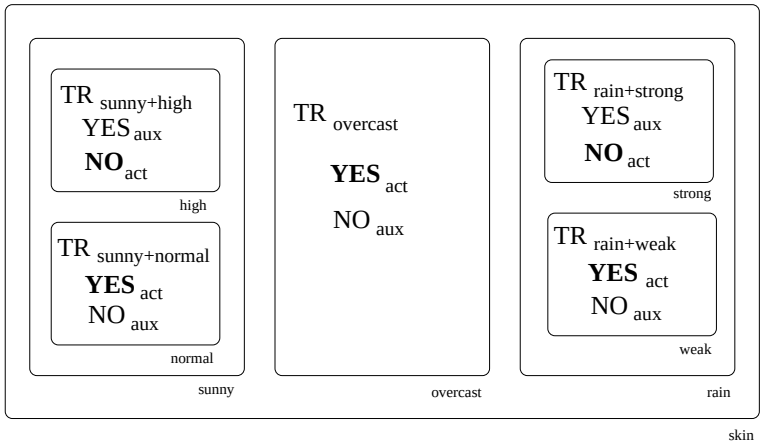


Fig. 3. Halting configuration for the self-constructed P system.

3.4 Theoretical Foundations

Next, we provide several considerations about the algorithm provided in the previous section. The first one is related to the training set provided as *input*. It must be carefully checked in order to avoid *noise*, i.e., it is not acceptable that the same instance appears with two different classifications in the same training set. Following the example from Table 1, we do not accept that two days with the same values for the attributes $Outlook$, $Temperature$, $Humidity$ and $Wind$ have different classifications for $PlayTennis$.

If the training set is free of noise, then the self-construction of the recognizer P system (first use) and the classification of new instances (second use) halt after a finite number of steps. A deeper question is the predictive power of the recognizer P system on new instances. A detailed study of such question is out of the scope of this paper. Nonetheless, let us briefly notice that usually, more than one decision tree is derivable from a training set and they may be not equivalent. From a Membrane Computing point of view, it is clear that the non-determinism produced by the set of rules **R3** produces a big amount of possible halting configurations.

As an illustrative example, let us consider a training set on wooden toy blocks with two attributes *Color* and *Shape* and the classification for *BelongsToEddy* shown in Table 2. Figure 4 shows two different trees consistent with the training set from Table 2. Both are *consistent*, since they classify correctly all the instances of the training set, but they are not equivalent. For example, the classification of a green and squared block, an instance which does not belongs to the training set, depends on the chosen tree. In Machine Learning, the followed criterion is *entia non sunt multiplicanda praeter necessitatem*, known as Ockham's razor[2] which states that among competing hypotheses, the one with the fewest assumptions

[2] Due to William of Ockham (1287 − 1347), see [8] for details.

Table 2. An example of training set for toy blocks.

Block	*Color*	*Shape*	*BelongsToEddy*
B1	Red	Square	Yes
B2	Red	Triangle	Yes
B3	Green	Triangle	No
B4	Green	Circle	No

should be selected. In this paper, we consider that all the trees consistent with the training set can be a solution of the problem and do not apply any *bias* for selecting a specific one.

The theoretical correctness, from a Machine Learning point of view, can be summed up in the following theorem.

Theorem 1. *Let us consider a non-empty training set D without noise. Let us consider that the instances in D has k attributes. Let Π be the self-constructing P system associated to D as shown above.*

(C1) *In the self-constructing stage, in any possible computation, there exists $p \in \{0, \ldots, k\}$ such that C_{1+5p} is a halting configuration. Therefore, each computation gives at most $1 + 5k$ steps.*

(C2) *Any computation in the decision stage gives $2p + 2$ steps with $p \in \{0, \ldots, k\}$ and sends to the environment YES_{act} or NO_{act} (but no both) in the last step of computation.*

(C3) *The recognizer P system obtained at the end of any computation of the self-constructing stage is* consistent *with the training set.*

The proof of this Theorem is provided in the Appendix A.

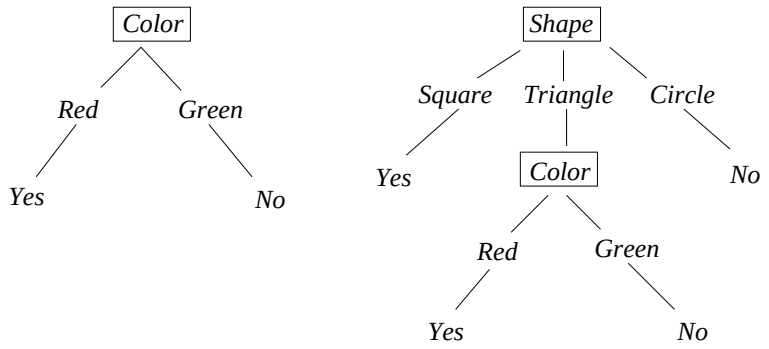


Fig. 4. Two trees consistent with the training set from Table 2. Notice that the classification for a green and squared block is different in both trees.

4 Conclusions

In this paper, we consider the process of modifying the membrane structure of the membranes as a target itself. This idea can be found in the literature (see, e.g., [3, 6]) but in this paper proposes a new point of view. The new membrane structure (potentially complex), which has been built according P system rules, is the initial membrane structure for a new computation. In this sense, we talk about *self-construction*. This self-construction of P systems can be adapted to many different scenarios and it is independent of the P system model. In such way, this proposal open a new research line, since the adaptation of this idea to different Membrane Computing models need a deeper study.

As an illustrative example, we have considered the well-known problem of constructing decision trees and their use as classifiers in the framework of Membrane Computing. This is also a contribution of this paper, since it provides a new bridge between Membrane Computing and Machine Learning. The purpose has been to illustrate the self-construction of P systems with a simple encoding that allows to translate easily the ideas from Machine Learning into Membrane Computing. In this way, the chosen codification allows a simple description of the process but it is far from being the most efficient. Different codifications will allow more efficient computations and further research must be done in this way.

Acknowledgements

MAGN acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

1. Ferretti, C., Mauri, G., Zandron, C.: P systems with string objects. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 168 – 197. Oxford University Press, Oxford, England (2010)
2. Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol. 3850. Springer, Berlin Heidelberg (2006)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Fractals and P systems. In: Graciani, C., Păun, Gh., Romero-Jiménez, A., Sancho-Caparrini, F. (eds.) Fourth Brainstorming Week on Membrane Computing. vol. II, pp. 65–86. Fénix Editora, Sevilla, Spain (2006)
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund et al. [2], pp. 224–240
5. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A linear solution for qsat with membrane creation. In: Freund et al. [2], pp. 241–252

6. Gutirrez-Naranjo, M.A., Perez-Jimnez, M.J., Riscos-Nez, A., Romero-Campero, F.J.: How to express tumours using membrane systems. Progress in Natural Science 17(4), 449–457 (2007)
7. Luisi, P.: The chemical implementation of autopoiesis. In: Fleischaker, G., Colonna, S., Luisi, P. (eds.) Self-Production of Supramolecular Structures, NATO ASI Series, vol. 446, pp. 179–197. Springer Netherlands (1994)
8. Mitchell, T.M.: Machine Learning. McGraw-Hill Education, 1st edn. (1997)
9. Murphy, N., Woods, D.: A characterisation of NL using membrane systems without charges and dissolution. In: Calude, C.S., Costa, J.F., Freund, R., Oswald, M., Rozenberg, G. (eds.) Unconventional Computing, Lecture Notes in Computer Science, vol. 5204, pp. 164–176. Springer Berlin Heidelberg (2008)
10. Mutyam, M., Krithivasan, K.: P systems with membrane creation: Universality and efficiency. In: Margenstern, M., Rogozhin, Y. (eds.) MCU. Lecture Notes in Computer Science, vol. 2055, pp. 276–287. Springer (2001)
11. Pérez-Jiménez, M.J., Riscos-Núñez, A.: A linear-time solution to the knapsack problem using P systems with active membranes. In: Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 2933, pp. 250–268. Springer, Berlin Heidelberg (2003)
12. Păun, G.: Computing with membranes. Tech. Rep. 208, Turku Centre for Computer Science, Turku, Finland (November 1998)
13. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000), see also [12]
14. Quinlan, J.R.: Induction of decision trees. Machine Learning 1(1), 81–106 (1986)
15. Song, T., Wang, X., Zheng, H.: Time-free solution to Hamilton path problems using P systems with d-division. Journal of Applied Mathematics 2013, Article ID 975798, 7 pages (2013)
16. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C., Dinneen, M.J. (eds.) UMC. pp. 289–301. Springer (2000)

A Appendix

Next, we provide the proof of the Theorem 1. In order to fix the notation, we will call *examples* to the strings $\overline{x}C$, where $\overline{c}$ is a string (maybe empty) of pairs *Attribute Value* and $C \in \{YES, NO\}$. First of all, we will prove the first statement:

> *In the self-constructing stage, in any possible computation, there exists* $p \in \{0, \ldots, k\}$ *such that* C_{1+5p} *is a halting configuration. Therefore, each computation gives at most* $1 + 5k$ *steps.* **(C1)**

In this self-construction stage the strings $\overline{x}?$ are not supplied to the system and, hence YES_{out} or NO_{out} are not produced. In this way, we do not care about the set of rules **R6**, **R7** and **R8** because they cannot be applied. The proof is based on the following lemmas.

Lemma 1. *Let us consider an elementary membrane at the step n such that there are no strings in its father membrane (it can also be the skin in the initial configuration) and its content is YES_{aux}, NO_{aux}, and a non-empty set of examples such that all of them end with YES or all of them end with NO. Then, the computation in this membrane finishes at step $n+1$.*

Proof. Let us consider that all the examples end with YES (the case NO is analogous). In the described conditions, only one rule can be applied (from **R1**) and in the step $n+1$, the membrane contains the same set of examples and YES_{act} and NO_{aux}. It is easy to check that no more rules can be applied and the computation finishes at this step.

From this lemma, we obtain that if all the examples in the initial set provided to the skin in the initial configuration end with YES or NO, then the computation of the self-constructing stage end after one computation step.

Lemma 2. *All the examples inside a membrane have the same length.*

Proof. It is trivial to check that it is true in the initial configuration, since if the training set has k attributes, then, all the initial examples have length $2k+1$. For the next steps, we only need to consider that the rules which sends examples from one membrane into other belongs tho the set **R5**, the application of these rules always decreases the length of the example in two units and all the examples arrive to the membrane simultaneously.

Lemma 3. *Let us consider an initial training set without noise. If in a membrane there is at least one example of length 1, then all of them are YES or all of them are NO.*

Proof. By, Lemma 2, we can consider that all the examples in the membrane have length 1. If the training set has k attributes, then all the examples in the initial configuration have length $2k+1$ and, by construction, these examples of length 1 came from these original one after deleting two objects k times. In this process, if two examples are sent to the same membrane, then both share the same value for one attribute. If both examples are in the same membrane after k deletions, then they share the values of the k attributes and, since their no noise, the classification must be the same.

Lemma 4. *Let us consider an elementary membrane h at the step n such that there are no strings in its father membrane (it can also be the skin in the initial configuration) and its content is YES_{aux}, NO_{aux}, and a non-empty set of examples of length l such that at least one of them ends with YES and at least one of them ends with NO. Then, at step $n+5$, the membrane h does not contains strings. It only contains elementary membranes such that contain YES_{aux}, NO_{aux} and examples of length $l-2$.*

Proof. In the described conditions, at the step $n+1$ the membrane contains the set of examples plus YES_{act} and NO_{act} (by **R1**); at the step $n+2$, it contains the set of examples and *new* (by **R2**); at $n+3$, it contains the set of examples plus $v_1,\ldots,v_s$, where $v_1,\ldots,v_s$ are the values of one of the attributes (by **R3**). By application of rules from **R4**, at the step $n+4$, the membrane contains the set of examples plus s elementary membranes, one for each value. Each of these membranes contains the strings YES_{aux} and NO_{aux}. Finally, rules from **R5** are applied and all the examples from the membrane h are sent into the elementary membranes by deleting two objects.

Finally, the Statement 1, can be proved from these lemmas.

Proof. Proof of the Statement 1. If the training set has k attributes, then the examples placed in the skin in the initial configuration have length $2k+1$. Two cases are possible:

- If all of them have the same classification, i.e., all of them end with YES or all of them end with NO, then, by Lemma 1, the computation finishes in the configuration C_1.
- If all of them do not have the same classification, then the conditions of Lemma 2 hold and the configuration C_5 has elementary membranes with YES_{aux}, NO_{aux} and examples of length $2(k-1)-1$. Each of these membranes is in the same conditions that the skin in the initial configuration, but the length of the examples has decreased in two units. This process goes on and each elementary membrane stops after $1+5p$ steps with $p \in \{0,\ldots,k\}$. Lemma 3 is considered to ensure that all the examples in the elementary membrane end in YES or NO and then, the computation halts as shown in Lemma 1.

Next, we will prove the second statement of the theorem. Now the P system has been self-constructed. Only elementary membranes have strings. As shown in Lemma 1, in the halting configuration, these membrane have a set of examples and the pair YES_{aux} and NO_{act} or YES_{act} and NO_{aux}, depending of the classification of the examples. Any computation in the decision stage starts by placing a string $\overline{x}?$ as input in the skin, where $\overline{v}$ is a string of pairs *Attribute Value*. Since all the examples in each membrane have the same classification and no more examples are supplied, then the rules from sets **R1** to **R5** cannot be applied. Only rules from **R6**, **R7** and **R8** must be considered. The statement is the following

Any computation in the decision stage gives $2p+2$ steps with $p \in \{0,\ldots,k\}$ and sends to the environment YES_{act} or NO_{act} (but no both) in the last step of computation. **(C2)**

Proof. First of all, let us notice that rules from **R6**, **R7** and **R8** cannot be applied simultaneously, because the conditions of applicability are disjoint and only one string $\overline{x}?$ is provided as input in the skin in each computation. From this observation, rules from **R7** are firstly applied p times with $p \in \{0,\ldots,k\}$. After these p steps, the elementary membrane where YES_{act} or NO_{act} is placed and in the

step $p+1$, the object YES_{act} or NO_{act} (but no both) is produced. After p steps more, in the step $2p+1$, YES_{act} or NO_{act} arrives to the skin by application of rules from **R8**. In the following step, $2p+2$, the object YES_{act} or NO_{act} is sent out to the environment and the computation halts.

In the third statement, we consider the self-constructed P system and take an example from the training set $\overline{x}C$, with $C \in \{YES, NO\}$ and replace C by ?. The statement claims that if we provide $\overline{x}$? to the P system in the decision stage, we obtain the same classification that the original one. The statement is

The recognizer P system obtained at the end of any computation of the self-constructing stage is consistent *with the training set.* **(C3)**

Proof. In order to fix ideas, let us consider that the original example was $\overline{x}YES$. The other case is analogous. By rules from **R6**, $\overline{x}$? will be sent to an elementary membrane where all the examples have the same classification. One of these examples was originally the $\overline{x}YES$ and YES_{act} occurs in this elementary membrane. Then, by application of one rule from **R7** and later with rules from **R8**, the object YES_{out} is sent to the environment.

Antimatter as a Frontier of Tractability in Membrane Computing

Daniel Díaz-Pernil[1], Francisco Peña-Cantillana[2],
Miguel A. Gutiérrez-Naranjo[2]

[1]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics - University of Sevilla, 41012, Spain
sbdani@us.es

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
frapencan@gmail.com, magutier@us.es

Summary. It is well known that the polynomial complexity class of recognizer polarizationless P systems with active membranes, without dissolution and with division for elementary and non-elementary membranes is exactly the complexity class **P** (see [6], Th. 2). In this paper, we prove that if such P system model is endowed with antimatter and annihilation rules, then **NP** problems can be solved. In this way, antimatter is a frontier of tractability in Membrane Computing.

1 Introduction

Antimatter is material composed of antiparticles, which have the same mass as particles of ordinary matter but have opposite charge. Encounters between particles and antiparticles lead to the annihilation of the objects, giving energy proportional to the total matter and antimatter mass, in accord with the mass-energy equivalence equation, $E = mc^2$.

The term antimatter was first used by Arthur Schuster in 1898, (see [15]). He hypothesized antiatoms, as well as whole antimatter solar systems, and discussed the possibility of matter and antimatter annihilating each other. The modern theory of antimatter began in 1928, with the papers [4, 5] by Paul Dirac. Dirac realised that the relativistic version of the Schrödinger wave equation for electrons predicted the possibility of antielectrons. These were discovered by Carl D. Anderson in 1932 [1] and named positrons (a contraction of "positive electrons").

In Membrane Computing, the notion of antimatter has been previously associated to anti-spikes in the framework of Spiking Neural P Systems (see, e.g., [10, 11, 16, 18]). In this context, when a spike and anti-spike appear in the same neuron, the annihilation occurs and both, spike and anti-spike, disappear.

In this paper, we prove that antimatter is a frontier of tractability in Membrane Computing. As detailed below, it is well known that the polynomial complexity class of recognizer P systems with active membranes without polarizations, without dissolution and with division of elementary and non-elementary division is exactly the complexity class **P** (see [6], Th. 2). In such paper, it is proved that if the described P system model is endowed with dissolution rules, then **NP**-complete problems can be solved. In this way, dissolution is a frontier of tractability.

In this paper, we consider the polynomial complexity class of recognizer P systems with active membranes without polarizations, without dissolution and with division of elementary and non-elementary division (i.e., the class which is equal to **P**) and we add antimatter and the corresponding annihilation rules. In this new model, we show a semi-uniform family of P systems which solves the Subset Sum problem. Since the Subset Sum Problem is **NP**-complete, this P system family shows that antimatter is a new frontier of the tractability in Membrane Computing

The paper is organized as follows: In the next section, we recall some basics on recognizer P systems, complexity classes and a previous result about dissolution as a frontier of tractability. Section 3 is devoted to the concept of antimatter in Membrane Computing. In Section 4, a solution to the Subset Sum problem by using antimatter and annihilation rules is shown. The paper finishes with some conclusions.

2 Recognizer P Systems

First of all, we recall the main notions related to recognizer P systems and complexity in Membrane Computing. For a detailed description, see, e.g., [12, 14].

The main *syntactic* ingredients of a cell–like P system are the *membrane structure*, the *multisets*, and the *evolution rules*. A *membrane structure* consists of several membranes arranged hierarchically inside a main membrane (the *skin*). Each membrane identifies a region inside the system. When a membrane has no membrane inside, it is called *elementary*. The objects can be described by symbols or by strings of symbols, in such a way that *multisets of objects* are placed in the regions of the membrane structure. The objects can evolve according to given *evolution rules*, associated with the regions.

The *semantics* of the cell–like membrane systems is defined through a non–deterministic and synchronous model. A *configuration* of a cell–like membrane system consists of a membrane structure and a family of multisets of objects associated with each region of the structure. At the beginning, there is a configuration called the *initial configuration* of the system. In each time unit we can transform a given configuration in another configuration by applying the evolution rules to the objects placed inside the regions of the configurations, in a non–deterministic, maximally parallel manner (the rules are chosen in a non–deterministic way, and in each region all objects that can evolve must do it). In this way, we get *transitions* from one configuration of the system to the next one. A *computation* of the

system is a (finite or infinite) sequence of configurations such that each configuration –except the initial one– is obtained from the previous one by a transition. A computation which reaches a configuration where no more rules can be applied to the existing objects and membranes, is called a *halting computation*. The result of a halting computation is usually defined through the multiset associated with a specific output membrane (or the environment) in the final configuration.

Let us recall that a decision problem X is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (the elements are called *instances*) and θ_X is a predicate (a total Boolean function) over I_X. Let $X = (I_X, \theta_X)$ be a decision problem. A *polynomial encoding* of X is a pair (cod, s) of polynomial time computable functions over I_X such that for each instance $w \in I_X$, $s(w)$ is a natural number representing the *size* of the instance and $cod(w)$ is an multiset representing an encoding of the instance. Polynomial encodings are stable under polynomial time reductions

2.1 The P systems Model

A P system with active membranes without polarizations, without dissolution and with division of elementary and non-elementary division is a P system with Γ as working alphabet, with H as the finite set of labels for membranes, and where the rules are of the following forms:

(a) $[\, a \to u \,]_h$ for $h \in H$, $a \in \Gamma$, $u \in \Gamma^*$. This is an object evolution rule, associated with a membrane labelled with h: an object $a \in \Gamma$ belonging to that membrane evolves to a string $u \in \Gamma^*$.

(b) $a\,[\ \ ]_h \to [\, b \,]_h$ for $h \in H$, $a, b \in \Gamma$. An object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object.

(c) $[\, a \,]_h \to b\ [\ \ ]_h$ for $h \in H$, $a, b \in \Gamma$. An object is sent out from membrane labelled with h to the region immediately outside, possibly transformed into another object.

(d) $[\, a \,]_h \to [\, b \,]_h\ [\, c \,]_h$ for $h \in H$, $a, b, c \in \Gamma$. An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects.

(e) $[\,[\,]_{h_1}[\,]_{h_2}\,]_{h_0} \to [\,[\,]_{h_1}\,]_{h_0}\,[\,[\,]_{h_2}\,]_{h_0}$, where h_0, h_1, h_2 are labels. They are division rules for non–elementary membranes. If the membrane with label h_0 contains other membranes than those with labels h_1, h_2, then such membranes and their contents are duplicated and placed in both new copies of the membrane h_0; all membranes and objects placed inside membranes h_1, h_2, as well as the objects from membrane h_0 placed outside membranes h_1 and h_2, are reproduced in the new copies of membrane h_0.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non–deterministic way), but any object which can evolve by one rule of any form, must evolve.

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

We denote by $\mathcal{AM}^0_{-d,+ne}$ the class of all recognizer P systems with active membranes without polarizations, without dissolution and with division of elementary and non-elementary division. We keep the subscript $-d$ in order to stress that no dissolution rules are used in this model.

2.2 Polynomial complexity classes in recognizer P systems

Let $\mathbf{\Pi} = (\Pi(w))_{w\in I_X}$ be a family of recognizer membrane systems and let $\mathcal{R}$ be a class of recognizer P systems without input membrane. A decision problem $X = (I_X, \theta_X)$ is solvable in a semi-uniform way and in polynomial time by the family $\mathbf{\Pi} = (\Pi(w))_{w\in I_X}$ of P systems of type $\mathcal{R}$, and we denote this by $X \in \mathbf{PMC}^*_{\mathcal{R}}$, if $\mathbf{\Pi}$ is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(w)$ from the instance $w \in I_X$; and $\mathbf{\Pi}$ is polynomially bounded, that is, there exists a polynomial function $p(n)$ such that for each $w \in I_X$, all computations of $\Pi(w)$ halt in at most $p(|w|)$ steps. It is said that $\mathbf{\Pi}$ is sound with regard to X if for each instance of the problem $w \in I_X$, if there exists an accepting computation of $\Pi(w)$, then $\theta_X(w) = 1$ and $\mathbf{\Pi}$ is complete with regard to X if for each instance of the problem $w \in I_X$, if $\theta_X(w) = 1$, then every computation of $\Pi(w)$ is an accepting computation.

Let $\mathcal{R}$ be a class of recognizer P systems with input membrane. A decision problem $X = (I_X, \theta_X)$ is solvable in a uniform way and polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$, of P systems from $\mathcal{R}$, and we denote this by $X \in \mathbf{PMC}_{\mathcal{R}}$, if the family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, i.e., there exists a polynomial encoding[1] (cod, s) from I_X to $\mathbf{\Pi}$ such that the family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps; and the family $\mathbf{\Pi}$ is sound and complete with regard to (X, cod, s). It is easy to see that the classes $\mathbf{PMC}^*_{\mathcal{R}}$ and $\mathbf{PMC}_{\mathcal{R}}$ are closed under polynomial–time reduction and complement.

According to these formal definitions, in [6] it is proved that the polynomial complexity class of recognizer P systems with active membranes without polarizations, without dissolution and with division of elementary and non-elementary division is exactly the complexity class $\mathbf{P}$. With the standard notation, $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0_{-d,+ne}} = \mathbf{PMC}^*_{\mathcal{AM}^0_{-d,+ne}}$.

[1] See [12, 14] for the details.

3 Antimatter

In this paper, we will use the physical inspiration of antimatter in the framework of cell-like P systems. In such way, given two object a and b from the alphabet Γ, an annihilation rule of a and b is written as $[ab \rightarrow \lambda]_h$. The *meaning* of the rule follows the idea of annihilation: If a and b occur simultaneously in the same membrane with label h, then both are consumed (disappear) and nothing is produced (denoted by the empty string λ). Let us remark that both objects a and b are objects from Γ and they can trigger any other rule of type (a) - (d) described above, not only annihilation rules. Nonetheless, in order to make the readability easier, if b annihilates the object a then b will be called the *antiparticle* of a and we will write $\overline{a}$ instead of b.

With respect to the semantics, let us notice that this rule must be applied as many times as possible in each membrane, according to the maximal parallelism, i.e., if m copies of a and n copies of $\overline{a}$ occur simultaneously in a membrane of label h, with $m \geq n$ and the rule $[a\overline{a} \rightarrow \lambda]_h$ is defined in the P system, then the rule is applied n times, n copies of a and $\overline{a}$ are consumed and $m - n$ copies of a are not affected by this rule.

Finally, a last consideration about the application of the annihilation rule. According to the non-determinism, if an object a can trigger more than one rule of types (a) - (d), then one rule of the applicable ones is non-deterministically chosen. Nonetheless, annihilation rules introduce a novelty. According to the physical intuition, if a and $\overline{a}$ occur simultaneously in the same membrane h and the annihilation rule $[a\overline{a} \rightarrow \lambda]_h$ is defined, then it is applied, regardless other options. In this sense, any annihilation rule has priority on the other types of rules.

For example, let us consider the rules $R_1 \equiv [a \rightarrow cd]_h$, $R_2 \equiv [a]_h \rightarrow b\,[\,]_h$ and $R_3 \equiv [a\overline{a} \rightarrow \lambda]_h$ defined on a membrane with label e. If the multiset inside the membrane is ab, then R_1 and R_2 are applicable and one of then must be non-deterministically chosen. Nevertheless, if the multiset is $ab\overline{a}$, then R_3 must be applied since it is an annihilation rule. If the multiset is $a^3b\overline{a}^2$, then the annihilation rule is applied twice, by consuming two copies of a and $\overline{a}$ and the third a is consumed by one rule of R_1 or R_2 non-deterministically chosen.

Formally, a P systems with active membranes, without polarizations, without dissolution, with division of elementary and non-elementary membranes and with annihilation rules is a construct of the form $\Pi = (O, H, \mu, w_1, \ldots, w_m, R)$ where:

1. $m \geq 1$ is the initial degree of the system;
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure* consisting of m membranes labelled with elements of H;
5. $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
6. R is a finite set of *rules* of the types (a) - (e) describe in the Section 2.1 and the following type of rules:

(f) $[\, a\overline{a} \rightarrow \lambda\,]_h$ for $h \in H$, $a\,\overline{a} \in \Gamma$. This is an annihilation rule, associated with a membrane labelled with h: the pair of objects $a\,\overline{a} \in \Gamma$ belonging simultaneously to that membrane disappear.

By following with the standard notation, we denote by $\mathcal{AM}^0_{-d,+ne,+ant}$ the class of these P systems, where $-d$ denotes that dissolution rules are not used, $+ne$ denotes the use of elementary and non elementary division and we add $+ant$ to denote the use of antimatter and annihilation rules.

4 Solving the Subset Sum Problem

In this paper we show that the class of decision problems solvable in polynomial time in a semi–uniform way by families of recognizer P systems in $\mathcal{AM}^0_{-d,+ne,+ant}$ contains the standard complexity class **NP**. Formally, we will prove the following theorem

Theorem 1. $\mathbf{NP} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0_{-d,+ne,+ant}}$

We will prove it by the construction of a semi-uniform family of such P systems that solves the Subset Sum Problem in a linear time. It is well known that the Subset Sum problem is the following one: *Given a finite set A, a weight function, $w : A \rightarrow \mathbb{N}$, and a constant $k \in \mathbb{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$.* This problem has been widely studied in Membrane Computing (see, e.g., [2, 3, 7, 8, 9, 13, 17]).

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

As usual, the idea of the design is better understood if we divide the solution to the problem into several stages:

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

For each instance $u = (n, (w_1, \ldots, w_n), k)$ of the Subset Sum problem we consider the P system $\Pi(u)$ defined as follows:

• Working alphabet:

$$\begin{aligned}\Gamma(u) = &\{a, \overline{a}, b, c_1, c_2, \overline{c}_2, d_0, \ldots, d_{2n+3}, e_1, \ldots, e_n\} \\ &\cup \{p_1, \ldots, p_n, k_{2n+4}, k_{2n+5}, r_{2n+2}, \ldots, r_{2n+6}\} \\ &\cup \{yes_{2n+8}, \ldots, yes_{2n+k+12}, \overline{yes}_{2n+8}, \overline{yes}_{2n+k+12}\} \\ &\cup \{z_{2n+4}, \ldots, z_{2n+7}, no_1, \ldots, no_{2n+k+10}, \overline{no}_{2n+k+10}, \overline{no}_{2n+k+11}\} \\ &\cup \{yes, no\}\end{aligned}$$

- Initial membrane structure: $\mu = [\,[\,[\,[\,]_0\,]_1\,]_2\,]_3$.
- Initial Multisets: $w_0 = d_0$, $w_1 = \emptyset$, $w_2 = \emptyset$ and $w_3 = no_0$.
- The set of evolution rules, $R(u)$, consists of the following rules.

Generation stage

(a) $[d_{2i} \to p_{i+1} d_{2i+1}]_0$ for $i \in \{0, \ldots, n-1\}$
$[d_{2i+1} \to d_{2i+2}]_0$ for $i \in \{0, \ldots, n-1\}$
$[d_{2n} \to d_{2n+1}]_0$
$[d_{2n+1} \to d_{2n+2} r_{2n+2}]_0$

The goal of the counter d_i is to control the apparition of the object p_i only in the odd steps. These p_i will produce the division of elementary membranes.

(b) $[p_i]_0 \to [e_i]_0\,[b]_0$ for $i \in \{1, \ldots, n\}$

The object p_i triggers the rule for division of elementary membranes: in one membrane is placed the object e_i and in the other the object b.

(c) $[\,[\,]_i\,[\,]_i\,]_{i+1} \to [\,[\,]_i\,]_{i+1}\,[\,[\,]_i\,]_{i+1}$ for $i \in \{0, 1, 2\}$.

This is the set of rules for the division of non-elementary membranes.

These three first set of rules produce the membrane structure needed for computing the solution. Notice that the objects p_i produce the division of the elementary membranes for $i \in \{1, \ldots, n\}$ and therefore, in the configuration C_{2i} there are 2^i elementary membranes. Let us also remark that the division of the elementary membranes is propagated by the rules of division on non elementary membranes, so in the configuration C_{2n+2}, the skin, labelled by 3, contains 2^n membranes labelled by 2. Each of them contains one membrane labelled by 1, and each of these membranes labelled by 1 contains one elementary membrane labelled by 0.

Weight calculation stage

(d) $[e_i \to a^{w_i}]_0$ for $i \in \{1, \ldots, n\}$

After the application of the membrane division rule by the object p_i, in one membrane is placed the object e_i and in the other the object b. Since the division is produced by p_i with $i \in \{1, \ldots, n\}$ this means that each of the 2^n elementary membranes receive a possible subset of $\{e_1, \ldots, e_n\}$. Object b remain inactive whereas the objects e_i evolve in the next step to as many objects s as the weight w_i. In such way, each elementary membrane contains as many objects s as the weight of the associated subset.

(e) $[d_{2n+2} \to d_{2n+3}]_0$
$[d_{2n+3} \to \overline{a}^{k+1}]_0$

When the generation stage has finished, the object d_{2n+3} produces $k+1$ copies of the object $\overline{a}$. These objects will interact with the object a by producing annihilation according to the following rule.

(f) $[a\overline{a} \to \lambda]_0$
$[\overline{a}]_0 \to b\,[\,]_0$

These are the key rules this stage. The rules $[e_i \to s^{w_i}]_0$ from the set b have generated as many copies of objects a as the weight of the subset encoded in the membrane. On the other hand, the rule $[d_{2n+3} \to \overline{a}^{k+1}]_0$ has generated $k+1$ copies of the object $\overline{a}$.

- If the weight of the subset encoded in the membrane (number of objects a) is greater than or equal to $k+1$ (number of objects $\overline{a}$), then all the objects $\overline{a}$ are consumed by the annihilation rules.
- If the weight of the subset encoded in the membrane (number of objects a) is equal to k, then the annihilation rule is applied k times and k copies of a and $\overline{a}$ are consumed. The remaining copy of $\overline{a}$ triggers the rule $[\overline{a}]_0 \to b\,[\,]_0$ and one object b appears in the corresponding membrane 1 in the configuration C_{2n+5}.
- Finally, if the weight of the subset encoded in the membrane (number of objects a) is lower than k, then all the copies of a are consumed by the annihilation rule, but p objects $\overline{a}$ are not affected by this rule, where $p \in \{2, \ldots, k+1\}$. These objects will trigger the rule $[\overline{a}]_0 \to b\,[\,]_0$, by due to the semantics of the P systems with active membranes, only one object can cross the membrane in each step and therefore, one object b will appear in the membrane 1 at the configuration C_{2n+5} whereas $p-1$ copies of $\overline{a}$ remain in the membrane 0.

(g) $[r_{2n+2}]_0 \to r_{2n+3}\,[\,]_0$
$[r_{2n+3} \to r_{2n+4} k_{2n+4} z_{2n+4}]_1$
$[r_{2n+4}]_1 \to r_{2n+5}\,[\,]_1$
$[k_{2n+4} \to k_{2n+5}]_1$
$[z_{2n+4} \to z_{2n+5}]_1$

The object r_{2n+2} produced by the last rule of the set (a) is the starting point for this set of rules. Its purpose is to place the counters r_i, z_i and k_i in the right membranes before starting the checking stage. Notice that the starting indices have been chosen for improving the readability. In this configuration C_{2n+5}, an object z_{2n+5} and an object k_{2n+5} are placed in each membrane with label 1 and one object r_{2n+5} is placed on each membrane with label 2.

Checking stage

This stage is quite technical. The aim is that each elementary membrane produces an object yes_{2n+k+9} in a membrane labelled by 2 in the configuration

C_{2n+k+9} if and only if in the configuration C_{2n+5} one and only one object $\overline{a}$ is placed in the elementary membrane.

(h) $[b \to c_1\overline{c}_2]_1$ $\quad$ $[k_{2n+5} \to c_2]_1$
$[c_1 \to c_2]_1$ $\quad$ $[z_{2n+5} \to z_{2n+6}]_1$
$[c_2\overline{c}_2 \to \lambda]_1$ $\quad$ $[z_{2n+6} \to z_{2n+7}]_1$
$[c_2]_1 \to yes_{2n+8}\,[\,]_1$ $\quad$ $[z_{2n+7} \to \overline{c}_2]_1$

Each object $\overline{a}$ in the elementary membrane is sent out, step by step, transformed into a b object. Since there exists one elementary membrane where the empty subset is encoded with zero objects a, then such elementary membrane contains $k+1$ objects $\overline{a}$ and the process of sending out these objects will take $k+1$ steps. Notice also that a new annihilation process (of the objects c_2 and $\overline{c}_2$) is also considered.

(i) $[r_{2n+5} \to r_{2n+6}]_2$ $\quad$ $[\overline{yes}_{2n+8} \to b]_2$
$[r_{2n+6} \to \overline{yes}_{2n+8}]_2$ $\quad$ $[yes_{2n+8}\,\overline{yes}_{2n+8} \to \lambda]_2$

The counter r_i produces the object $\overline{yes}_{2n+8}$ in the membrane 2 exactly in the configuration C_{2n+7}. If in this time there is an object yes_{2n+8} in the membrane, both are annihilated. If not, $\overline{yes}_{2n+8}$ evolves to b. The purpose is this set of rules is to control the evolution of the object yes_{2n+8}. If it appears in a membrane labelled by 2 in the configuration C_{2n+7}, it will be annihilated. If it appears later, it will survive.

(j) $[yes_{2n+8+i} \to yes_{2n+8+(i+1)}]_2$ for $i \in \{0, \ldots, k-1\}$

The objects yes_i evolves in the membrane 2 waiting for the end of this stage. As pointed above, the $k+1$ objects $\overline{a}$ from the membrane encoding the empty subset take $k+1$ steps to cross out the membrane.

The result of the checking stage can be summarized in the following lemma.

Lemma 1. *In each of the 2^n membrane structures $[\,[\,[\,]_0\,]_1\,]_2$ at configuration C_{2n+k+8}:*

- *Membranes 0 and 1 are inactive. No rules can be applied inside them.*
- *Membrane 2 contains one object yes_{2n+k+8} if and only if the number of objects a in the membrane 0 at the configuration C_{2n+4} is exactly k. In other words, if it corresponds to a subset of weight k.*

This lemma will be proved in the Appendix.

Output stage

(k) $[no_i \to no_{i+1}]_3$ for $i \in \{0, \ldots, 2n+k+9\}$
$[no_{2n+k+10}]_3 \to no\,[\,]_3$

From the initial configuration, the counter no_i is evolving[2]. If the evolution is not interrupted, an object no is sent out as answer of the computation. In

[2] Of course, this counter also evolves in the previous stages, but it has not been mentioned for the sake of simplicity.

this design, if an object yes_{2n+k+8} occurs in a membrane labelled by 2, then the counter no_i is stopped and yes will be sent out as the answer. Nonetheless, more than one of such objects yes_{2n+k+8} can be produced in different membranes. Dealing with this possibility needs to add some technical rules.

(l)
$[yes_{2n+k+8}]_2 \to yes_{2n+k+9}\,[\,]_2$
$[yes_{2n+k+9} \to \overline{no}_{2n+k+10}\,yes_{2n+k+10}]_3$
$[yes_{2n+k+10} \to yes_{2n+k+11}]_3$
$[yes_{2n+k+11} \to yes_{2n+k+12}]_3$
$[yes_{2n+k+12}\,\overline{yes}_{2n+k+12} \to \lambda]_3$
$[yes_{2n+k+12}]_3 \to yes\,[\,]_3$

This set of rules, together with the next one, controls the output of the system. The key ideas are that the object yes_{2n+k+9} produces an object $\overline{no}_{2n+k+10}$, which stops the counter no_i and $yes_{2n+k+12}]_3$ sends out the answer yes in the last step of computation.

(m)
$[no_{2n+k+9} \to no_{2n+k+10}]_3$
$[no_{2n+k+10}]_3 \to no\,[\,]_3$
$[no_{2n+k+10}\overline{no}_{2n+k+10} \to \lambda]_3$
$[\overline{no}_{2n+k+10} \to \overline{no}_{2n+k+11}]_3$
$[\overline{no}_{2n+k+11} \to \overline{yes}_{2n+k+12}]_3$

This is the last set of rules in our design. Let us remark that the object $no_{2n+k+11}$ sends to the environment the object no as an answer if no object $\overline{no}_{2n+k+11}$ is produced. If this object is produced, then the annihilation occurs and the object no is never sent out. The result of this checking stage is summed up in the following lemma.

Lemma 2. *If any of the 2^n membranes with label 2 contains an object yes_{2n+k+8} in the configuration C_{2n+k+8}, then the P system halts at the configuration $C_{2n+k+13}$ and sends yes to the environment in the last step of computation. Otherwise, the P system halts at the configuration $C_{2n+k+11}$ and sends no to the environment in the last step of computation.*

This lemma will be proved in the Appendix and it finishes the proof of Th. 1.

5 Conclusions

In this paper, we present a new frontier of tractability in Membrane Computing by adding annihilation rules and the concept of antimatter to a P system model with active membranes with division and without dissolution. Let us remark that the presented design makes use of a singularity in the semantics of the annihilation rule. According to the physical intuition, in presence of the antiparticle $\overline{a}$, the particle a has no option and both are annihilated. The translation of this intuition is a priority relation with respect to the remaining applicable rules. An open problem is to know if removing this priority feature from the model, it is still possible to solve **NP** problems.

Acknowledgements

MAGN acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

1. Anderson, C.D.: The positive electron. Physical Review 43, 491–494 (1933)
2. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A logarithmic bound for solving subset sum with P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 257–270. Springer, Berlin Heidelberg (2007)
3. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving subset sum in linear time by using tissue P systems with cell division. In: Mira, J., Álvarez, J.R. (eds.) IWINAC (1). Lecture Notes in Computer Science, vol. 4527, pp. 170–179. Springer, Berlin Heidelberg (2007)
4. Dirac, P.A.M.: The quantum theory of the electron. I. Proceedings of the Royal Society A 117(778), 610–624 (1928)
5. Dirac, P.A.M.: The quantum theory of the electron. II. Proceedings of the Royal Society A 118(779), 351–361 (1928)
6. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 3850, pp. 224–240. Springer, Berlin Heidelberg (2005)
7. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A linear solution of subset sum problem by using membrane creation. In: Mira, J., Álvarez, J.R. (eds.) IWINAC (1). Lecture Notes in Computer Science, vol. 3561, pp. 258–267. Springer, Berlin Heidelberg (2005)
8. Leporati, A., Gutiérrez-Naranjo, M.A.: Solving subset sum by spiking neural P systems with pre-computed resources. Fundamenta Informaticae 87(1), 61–77 (2008)
9. Leporati, A., Mauri, G., Zandron, C., Păun, Gh., Pérez-Jiménez, M.J.: Uniform solutions to SAT and subset sum by spiking neural P systems. Natural Computing 8(4), 681–702 (2009)
10. Metta, V.P., Krithivasan, K., Garg, D.: Computability of spiking nueral P systems with anti-spikes. New Mathematics and Natural Computation (NMNC) 08(03), 283–295 (2012)
11. Pan, L., Păun, Gh.: Spiking neural P systems with anti-spikes. International Journal of Computers, Communications & Control IV(3), 273–282 (September 2009)
12. Pérez-Jiménez, M.J.: An approach to computational complexity in membrane computing. In: Mauri, G., Păun, Gh., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 3365, pp. 85–109. Springer (2004)
13. Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving the subset-sum problem by P systems with active membranes. New Generation Computing 23(4), 339–356 (2005)

14. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 302 – 336. Oxford University Press, Oxford, England (2010)
15. Schuster, A.: Potential Matter. A Holiday Dream. Nature 58, 367 (Aug 1898)
16. Song, T., Jiang, Y., Shi, X., Zeng, X.: Small universal spiking neural P systems with anti-spikes. Journal of Computational and Theoretical Nanoscience 10(4), 999–1006 (2013)
17. Song, T., Luo, L., He, J., Chen, Z., Zhang, K.: Solving subset sum problems by time-free spiking neural P systems. Applied Mathematics & Information Sciences 8(1), 327–332 (2014)
18. Tan, G., Song, T., Chen, Z., Zeng, X.: Spiking neural P systems with anti-spikes and without annihilating priority working in a 'flip-flop' way. International Journal of Computing Science and Mathematics 4(2), 152–162 (Jul 2013)

Appendix

Firstly, we will prove Lemma 1.

Proof. The proof is by inspection of the cases. In order to simplify the notation, we will omit the occurrences of objects a and b in the membrane 0, since they do not trigger any rule. The process is deterministic, so we will not remark the applied rules. They can be easily found in the corresponding set.

Case 1. Let us suppose that the weight of the encoded set is greater than k. In this case, the rule $[\overline{a}]_0 \to b\,[\,]_0$ is no applied and there are no objects $\overline{a}$ in membrane 0 and no object b in the membrane 1 at the configuration C_{2n+5}. With the considerations claimed above, the evolution of the membrane structure is

$$\begin{array}{ll}
C_{2n+5} & \equiv [\,[\,[\,]_0\, k_{2n+5}\, z_{2n+5}\,]_1\, r_{2n+5}\,]_2 \\
C_{2n+6} & \equiv [\,[\,[\,]_0\, c_2\, z_{2n+6}\,]_1\, r_{2n+6}\,]_2 \\
C_{2n+7} & \equiv [\,[\,[\,]_0\, z_{2n+7}\,]_1\, yes_{2n+8}\, \overline{yes}_{2n+8}\,]_2 \\
C_{2n+8} & \equiv [\,[\,[\,]_0\, \overline{c}_2\,]_1\,]_2 \\
\ldots & \ldots \\
C_{2n+k+8} & \equiv [\,[\,[\,]_0\, \overline{c}_2\,]_1\,]_2
\end{array}$$

In this case, the computation stops at the configuration C_{2n+8}. Since no more rules are applied, the result holds for C_{2n+k+8}.

Case 2. Let us suppose that the weight of the encoded set is exactly equal to k. In this case, in the configuration C_{2n+5} there are one object b in the corresponding membrane 1 and no objects in the membrane 0.

$$\begin{array}{ll}
C_{2n+5} & \equiv [\,[\,[\,\overline{a}\,]_0\, b\, k_{2n+5}\, z_{2n+5}\,]_1\, r_{2n+5}\,]_2 \\
C_{2n+6} & \equiv [\,[\,[\,]_0\, c_1\overline{c}_2\, c_2\, z_{2n+6}\,]_1\, r_{2n+6}\,]_2 \\
C_{2n+7} & \equiv [\,[\,[\,]_0\, c_2\, z_{2n+7}\,]_1\, \overline{yes}_{2n+8}\,]_2 \\
C_{2n+8} & \equiv [\,[\,[\,]_0\, \overline{c}_2\,]_1\, b\, yes_{2n+8}\,]_2 \\
\ldots & \ldots \\
C_{2n+k+8} & \equiv [\,[\,[\,]_0\, \overline{c}_2\,]_1\, b\, yes_{2n+k+8}\,]_2
\end{array}$$

Case 3. Let us suppose that the weight of the encoded set is lower than k. In this case, in the configuration C_{2n+5} there are $p-1$ objects $\overline{a}$ in the corresponding membrane 0 and one b in the membrane 1 with $p \in \{2, \ldots, k+1\}$. . We split this case into three subcases.

Case 3a: $p = 2$

$$\begin{array}{ll}
C_{2n+5} & \equiv [\,[\,[\,\overline{a}\,]_0\, b\, k_{2n+5}\, z_{2n+5}\,]_1\, r_{2n+5}\,]_2 \\
C_{2n+6} & \equiv [\,[\,[\,]_0\, b\, c_1 \overline{c}_2\, c_2\, z_{2n+6}\,]_1\, r_{2n+6}\,]_2 \\
C_{2n+7} & \equiv [\,[\,[\,]_0\, c_1\, \overline{c}_2\, c_2\, z_{2n+7}\,]_1\, \overline{yes}_{2n+8}\,]_2 \\
C_{2n+8} & \equiv [\,[\,[\,]_0\, c_2\, \overline{c}_2\,]_1\, b\,]_2 \\
C_{2n+9} & \equiv [\,[\,[\,]_0\,]_1\, b\,]_2 \\
\ldots & \ldots \\
C_{2n+k+8} & \equiv [\,[\,[\,]_0\,]_1\, b\,]_2
\end{array}$$

Case 3b: $p = 3$

$$\begin{array}{ll}
C_{2n+5} & \equiv [\,[\,[\,\overline{a}^2\,]_0\, b\, k_{2n+5}\, z_{2n+5}\,]_1\, r_{2n+5}\,]_2 \\
C_{2n+6} & \equiv [\,[\,[\,\overline{a}\,]_0\, b\, c_1 \overline{c}_2\, c_2\, z_{2n+6}\,]_1\, r_{2n+6}\,]_2 \\
C_{2n+7} & \equiv [\,[\,[\,]_0\, b\, c_1\, \overline{c}_2\, c_2\, z_{2n+7}\,]_1\, \overline{yes}_{2n+8}\,]_2 \\
C_{2n+8} & \equiv [\,[\,[\,]_0\, c_1\, c_2\, \overline{c}_2^2\,]_1\, b\,]_2 \\
C_{2n+9} & \equiv [\,[\,[\,]_0\, c_2\, \overline{c}_2\,]_1\, b\,]_2 \\
C_{2n+10} & \equiv [\,[\,[\,]_0\,]_1\, b\,]_2 \\
\ldots & \ldots \\
C_{2n+k+8} & \equiv [\,[\,[\,]_0\,]_1\, b\,]_2
\end{array}$$

Case 3c: $p \geq 4$

$$\begin{array}{ll}
C_{2n+4+1} & \equiv [\,[\,[\,\overline{a}^{p-1}\,]_0\, k_{2n+5}\, z_{2n+5}\,]_1\, r_{2n+5}\,]_2 \\
C_{2n+4+2} & \equiv [\,[\,[\,\overline{a}^{p-2}\,]_0\, b\, c_1 \overline{c}_2\, c_2\, z_{2n+6}\,]_1\, r_{2n+6}\,]_2 \\
C_{2n+4+3} & \equiv [\,[\,[\,\overline{a}^{p-3}\,]_0\, b\, c_1\, \overline{c}_2\, c_2\, z_{2n+7}\,]_1\, \overline{yes}_{2n+8}\,]_2 \\
C_{2n+4+4} & \equiv [\,[\,[\,\overline{a}^{p-4}\,]_0\, b\, c_1\, c_2\, \overline{c}_2^2\,]_1\, b\,]_2 \\
\ldots & \ldots \\
C_{2n+4+i} & \equiv [\,[\,[\,\overline{a}^{p-i}\,]_0\, b\, c_1\, c_2\, \overline{c}_2^2\,]_1\, b\,]_2 \\
\ldots & \ldots \\
C_{2n+4+(p-1)} & \equiv [\,[\,[\,\overline{a}\,]_0\, b\, c_1\, c_2\, \overline{c}_2^2\,]_1\, b\,]_2 \\
C_{2n+4+p} & \equiv [\,[\,[\,]_0\, b\, c_1\, c_2\, \overline{c}_2^2\,]_1\, b\,]_2 \\
C_{2n+4+p+1} & \equiv [\,[\,[\,]_0\, c_1\, c_2\, \overline{c}_2^2\,]_1\, b\,]_2 \\
C_{2n+4+p+2} & \equiv [\,[\,[\,]_0\, c_2\, \overline{c}_2\,]_1\, b\,]_2 \\
C_{2n+4+p+3} & \equiv [\,[\,[\,]_0\,]_1\, b\,]_2
\end{array}$$

Notice that in one of the elementary membranes, the empty set is encoded. It means that in each computation, in one of the 2^n membranes, $p = k+1$ (the greater value). In this case, $C_{2n+4+(k+1)+3} = C_{2n+k+8}$.

Next, we provide the proof of the Lemma 2.

Proof. The proof is also by inspection of the cases. As in the previous proof, the computations are deterministic, and we do not remark the applied rules.

Case 1: Let us consider that there exists exactly one membrane with label 2 contains an object yes_{2n+k+8} in the configuration C_{2n+k+8}. By application of the rule $[yes_{2n+k+8}]_2 \rightarrow yes_{2n+k+9}\,[\,]_2$, an object yes_{2n+k+9} arrives to the membrane 2 in the configuration C_{2n+k+9}. Next, we show the evolution of this membrane.

$$\begin{aligned}
C_{2n+k+9} &\equiv [\, yes_{2n+k+9}\, no_{2n+k+9}\,]_3\\
C_{2n+k+10} &\equiv [\, yes_{2n+k+10}\, \overline{no}_{2n+k+10}\, no_{2n+k+10}\,]_3\\
C_{2n+k+11} &\equiv [\, yes_{2n+k+11}\,]_3\\
C_{2n+k+12} &\equiv [\, yes_{2n+k+12}\,]_3\\
C_{2n+k+13} &\equiv yes\,[\,]_3
\end{aligned}$$

Case 2: Let us consider that there exist t membranes ($t \geq 2$) with label 2 containing an object yes_{2n+8} in the configuration C_{2n+k+8}. By application of the rule $[yes_{2n+k+8}]_2 \rightarrow yes_{2n+k+9}\,[\,]_2$, t objects yes_{2n+k+9} arrive to the membrane 2 in the configuration C_{2n+k+9}. Next, we show the evolution of this membrane.

$$\begin{aligned}
C_{2n+k+9} &\equiv [\, yes^{t}_{2n+k+9}\, no_{2n+k+9}\,]_3\\
C_{2n+k+10} &\equiv [\, yes^{t}_{2n+k+10}\, \overline{no}^{p}_{2n+k+10}\, no_{2n+k+10}\,]_3\\
C_{2n+k+11} &\equiv [\, yes^{t}_{2n+k+11}\, \overline{no}^{t-1}_{2n+k+11}\,]_3\\
C_{2n+k+12} &\equiv [\, yes^{t}_{2n+k+12}\, \overline{yes}^{t-1}_{2n+k+13}]_3\\
C_{2n+k+13} &\equiv yes\,[\,]_3
\end{aligned}$$

Case 3: Finally, let us consider that there do not exist membranes with label 2 containing an object yes_{2n+8} in the configuration C_{2n+k+8}.

$$\begin{aligned}
C_{2n+k+9} &\equiv [\, no_{2n+k+9}\,]_3\\
C_{2n+k+10} &\equiv [\, no_{2n+k+10}\,]_3\\
C_{2n+k+11} &\equiv no\,[\,]_3
\end{aligned}$$

P Systems with Anti-Matter

Rudolf Freund[1], Gheorghe Păun[2]

[1] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
rudi@emcc.at

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700, Bucharest, Romania
gpaun@us.es, curteadelaarges@gmail.com

Summary. After a short introduction to the area of membrane computing (a branch of natural computing), we introduce the concept of anti-matter in membrane computing. First we consider spiking neural P systems with anti-spikes, and then we show the power of anti-matter in cell-like P systems. As expected, the use of anti-matter objects and especially of matter/anti-matter annihilation rules, turns out to be rather powerful: computational completeness of P systems with anti-matter is obtained immediately, even without using catalysts. Finally, some open problems are formulated, too.

1 Introduction

First we give a brief introduction to membrane computing, a branch of natural computing having widely developed during the more than fifteen years since its initiation, see [19]. In some details we present a specific class of membrane systems (usually called *P systems*) with motivation coming from the way neurons interact, the spiking neural (in short, SN) P systems. In particular, we discuss SN P systems with *anti-spikes*, and then we generalize this idea, considering P systems of any type with *anti-objects*: for an object a, we say that $\bar{a}$ is an anti-object if an *annihilation rule* $a\bar{a} \to \lambda$ is assumed to exist in all membranes, which may either be an explicit rule or else act in an implicit way by removing a pair $a, \bar{a}$ in zero time. These annihilation rules turn out to be rather powerful, as somehow expected if, for example, we look at the λ-rules as the only non-context-free rules in the Geffert normal forms, e.g., see [22].

2 Elements of Membrane Computing

Membrane computing is a branch of natural computing, aiming to abstract computing models from the structure and the functioning of living cells. The models

obtained in that way are called *P systems*. Single cells (leading to *cell-like P systems*) as well as communities of cells, like tissues or organs (leading to t*issue-like P systems*), or neural cells (the associated models are called *spiking neural P systems*; these are one of the main topics in the present survey paper) have been considered in the literature. Basically, a P system consists of an arrangement of *membranes* (arranged in a hierarchical manner in the cell-like case and placed in the nodes of an arbitrary graph in the tissue-like case), which determine *compartments* where *multisets of objects* are placed, together with *evolution rules* inspired from biochemistry. Using these rules, the objects evolve, and these evolutions of objects are considered as *computations*. A result is associated with certain computations, hence, a computing device is obtained (working in the generative, the accepting, or the computing mode).

This very general framework lead to a large number of specific classes of P systems. Details can be found, for example, in [20] and [21]; recent information is available at the membrane computing website [27].

As objects in a P system we may use multisets of symbols from a given (finite) alphabet, strings, or more complex structures, such as graphs or d-dimensional arrays. In the case of spiking neural P systems, only multisets over a single object – the *spike*, an electrical impulse used by neurons to communicate with each other – are used. The rules used in a P system are of various types: multiset rewriting rules (similar to chemical reactions), string processing rules, specific rules for handling spikes, or rules directly inspired from biology, such as symport/antiport rules (for moving coupled symbol objects through membranes, corresponding to the functioning of selective protein channels in biology), or rules for handling membranes (dividing or creating membranes, exocytosis, endocytosis, etc.). The rules can be used sequentially or in parallel; the basic strategy in membrane computing is to use the rules in the maximally parallel way (in each step, a maximal multiset of rules is used in each compartment, in the multiset inclusion sense: no rule can be added to a chosen multiset of rules such that the resulting multiset of rules would still be applicable). Most of the investigations carried out in the literature concern synchronized P systems, but also non-synchronized systems were considered. In what concerns the ways to associate a result to a computation, there also are several possibilities: usually, only halting computations are considered to be successful (those computations which reach a configuration of the system where no rule can be applied any more). When dealing with multisets (which is also the case when dealing with SN P systems), the natural result of a computation is a number, but also strings can be associated in various ways.

The computing power of these devices is rather large: Turing computability can be obtained by many classes of P systems. In the cases when an exponential working space can be created during the computation in polynomial time, then, by a time-space trade-off, polynomial, often even linear, solutions to computationally hard problems (typically, **NP**-complete problems, but sometimes even **PSPACE**-problems) can be obtained.

Power and efficiency are computer science issues. Membrane computing proved to be rather attractive as a modeling framework, too. The reader can consult [3] and [4] in this respect. The most numerous and advanced applications are those in biology and biomedicine, but there are also well-investigated applications in approximate optimization, computer graphics, robot control, etc.

In this paper, we formally introduce only the basic model of membrane computing, the cell-like P systems with symbol objects (which will also be considered in Section 5 below). Hierarchical membrane structures (which can be described by a tree) are represented by strings of labeled matching parentheses, and the multisets over an alphabet V are represented by strings over V; a string and all its permutations represent the same multiset. For an alphabet V, by V^* we denote the set of all strings over V, including the empty string. The length of $x \in V^*$ is denoted by $|x|$; the empty string, of length zero, is denoted by λ. Basic knowledge in formal language theory as well as some familiarity with basic elements of membrane computing is assumed in what follows.

A *cell-like P system*, of degree m, with catalysts, is a construct

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{in}, i_{out})$$

where O is the alphabet of objects, $C \subset O$ is the set of catalysts, μ is the *membrane structure* (with m membranes), $w_1, \ldots, w_m$ are strings over O representing multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of evolution rules associated with the regions of μ, and i_{in} and i_{out} are the labels of the input and output regions, respectively; if the input or output is taken from the environment, this is indicated by taking the label 0 for i_{in} or i_{out}, respectively.

The *evolution rules* are multiset rewriting rules of the form $u \to v$, where u is a non-empty multiset over O and $v = (b_1, tar_1) \ldots (b_k, tar_k)$ with $b_i \in O$ and $tar_i \in \{here, out, in\}$, i.e., the objects b_i in v have associated a *target indication* tar_i. Using such a rule means "consuming" the objects of u and "producing" the objects from $b_1, \ldots, b_k$ of v, where the target indication *here* means that the objects remain in the same region where the rule is applied, *out* means that they are sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied in the skin region), and *in* means that they are sent to one of the immediately inner membranes, chosen in a non-deterministic way; in general, the target indication *here* is omitted.

A rule $u \to v$ with $|u| = 1$ is said to be *non-cooperative*. A rule of the form $ca \to cv$, where $c \in C$, $a \in V \setminus C$, and the objects in v are from $V \setminus C$, too, is called *catalytic*; C is the set of catalysts, objects which are not changed by evolution rules. Arbitrary rules are called *cooperative*.

If the system is used in the generative mode, then i_{in} is omitted, and if the system is used in the accepting mode, then i_{out} is omitted. The number m of membranes in μ is called the *degree* of Π.

In the generative case, the set of numbers computed by Π (in the maximally parallel non-deterministic mode) is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$

computed by systems Π of degree at most $m \geq 1$ and using rules of form α is denoted by $NOP_m(\alpha)$; if there is no bound on the degree of the systems, then the subscript m is replaced by $*$. According to the previous classification, $\alpha \in \{ncoo, cat, coo\}$, with the obvious meaning.

It is known that $NOP_1(coo) = NOP_1(cat_2) = NRE$, where cat_2 indicates the fact that only two catalysts are used with catalytic rules together with non-cooperative rules, and NRE is the family of recursively enumerable (Turing computable) sets of natural numbers. In turn, $NOP_*(ncoo) = NREG$, where $NREG$ is the family of length sets of regular languages (i.e., the family of semilinear sets of natural numbers).

3 Spiking Neural P Systems

Spiking neural P systems, see [8], have a completely different architecture and functioning, as they are not based on the standard eukaryotic cell, but on brain biology. Here we only consider the neurons cooperating by means of spikes, electrical impulses of identical forms, moving along axons. Spiking neurons are also investigated in the current neural computing area (e.g., see [11]). We do not define SN P systems in a formal way, instead we only describe such a system and then also introduce anti-spikes, and we will give a simple example.

In short, an SN P system consists of a set of *neurons* (represented by membranes) placed in the nodes of a directed graph (the arcs are called *synapses*) and containing *spikes*, denoted by copies of the symbol a. Thus, the architecture is that of a tissue-like P system, with only one kind of objects present in the cells. The objects evolve by means of *spiking rules*, which are of the form $(E/a^c \rightarrow a; d)$, where E is a regular expression over a and c, d are natural numbers, $c \geq 1$, $d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E)$, $k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \rightarrow \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes. The system works in a synchronized manner, i.e., in each time unit, every neuron which can use a rule should do that, but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output* one, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system; it might be infinite if the computation does not stop.

The result of a computation is encoded in the distance between the first two spikes sent to the environment by the (output neuron of the) system. Other ways to associate a result with a computation were considered, for instance, the total number of spikes emitted by the output neuron during a halting computation, or else the number of spikes contained in the output neuron at the end of a halting

computation; the spike train itself can be taken as the result of the computation, and in this way the system generates a binary sequence (a finite string, if the computation halts).

There are several classes of SN P systems, using various combinations of ingredients – rules of restricted forms, for example, without a delay, without forgetting rules, or extended rules, e.g., producing more than one spike, as well as asynchronous SN P systems (no clock is considered, any neuron may use a rule or not), with exhaustive use of rules (when enabled, a rule is used as many times as made possible by the spikes present in a neuron), with certain further conditions imposed on the halting configuration, with the same sets of rules in each neuron (the system then is called *homogeneous*), containing further biological ingredients, such as *astrocytes*, with inhibitory synapses, etc. For most SN P systems with unbounded neurons (arbitrarily many spikes can be found in each of them), characterizations of Turing computable sets of natural numbers are obtained. When the neurons are bounded, usually characterizations of the family $NREG$ are obtained. SN P systems can also be used in the accepting and in the computing modes.

4 SN P Systems with Anti-Spikes

A natural feature added to an SN P system is that of *anti-spikes*, proposed in [17] and then investigated in a series of papers. For the reader's convenience, the bibliography below contains many titles of papers dealing with this subject, yet not all of them are explicitly referred to in the present suvey paper.

The main point of the new notion is to interpret the "anti-spikes" as "anti-matter", hence to assume that when a piece of matter meets the corresponding piece of anti-matter, they will annihilate each other. This corresponds to the existence of rules of the form $a\bar{a} \to \lambda$, which are used immediately when a and $\bar{a}$ are present in the same neuron.

Thus, in an SN P system with anti-spikes, the spiking rules and the forgetting rules are of the forms $E/b^c \to b'^c$ and $b^c \to \lambda$ where E is a regular expression over a or over $\bar{a}$, while $b, b' \in \{a, \bar{a}\}$ and $c \geq 1$. If $L(E) = b^c$, then we write the first rule as $b^c \to b'$. As usual, a delay can be added to the spiking rules, too.

Note that we have four categories of rules, identified by $(b, b') \in \{(a, a), (a, \bar{a}), (\bar{a}, a), (\bar{a}, \bar{a})\}$. Of course, it is of interest to restrict the type of rules, and this is the case in most papers found in the literature.

The rules are used as usual in SN P systems, with the additional fact that a and $\bar{a}$ "cannot stay together", they instantaneously annihilate each other: if in a neuron there are either objects a or objects $\bar{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^r and $\bar{a}^s$ inside, then immediately the rule of the form $a\bar{a} \to \lambda$ is applied in the maximal manner, so that either the multiset of spikes a^{r-s} – if $r \geq s$ – or of anti-spikes $\bar{a}^{s-r}$ – if $s \geq r$ – remains.

In the definition from [17], the mutual annihilation of spikes and anti-spikes takes no time, so that the neurons always contain either only spikes or only anti-

spikes. That is why, for instance, the regular expressions of the spiking rules are defined either over a or over $\bar{a}$, but not over both symbols. Moreover, annihilation has priority over spiking and forgetting rules. Later, also the case when the annihilation takes one time unit was considered, with explicitely using the rule $a\bar{a} \to \lambda$, eventually even without priority over other rules.

The computations and the results of computations are defined in the same way as for usual SN P systems. In most investigations, the restriction was considered that the output neuron produces only spikes, not also anti-spikes. The anti-spikes are sometimes used to encode, in a natural way, negative numbers.

By $N_2(\Pi)$ we denote the family of numbers generated by an SN P system (with anti-spikes) as the distance between the first two spikes sent to the environment by the output neuron, and by $N_2S_aNP_m$ the families of all sets $N_2(\Pi)$, computed by SN P systems with anti-spikes and at most $m \geq 1$ neurons. When the number of neurons is not bounded, we replace the subscript m by $*$.

We illustrate the previous definition by an example recalled from [17]; it is, in fact, part of the proof showing computational completeness of SN P systems with anti-spikes (i.e., $N_2S_aNP_* = NRE$), namely, the module which simulates a SUB-instruction of a register machine. We present the module in the graphical form, a usual way of presentation in membrane computing: neurons are given as ovals containing spiking and forgetting rules, and in addition indicating the initial spikes and anti-spikes; the synapses are represented by arrows linking the neurons.

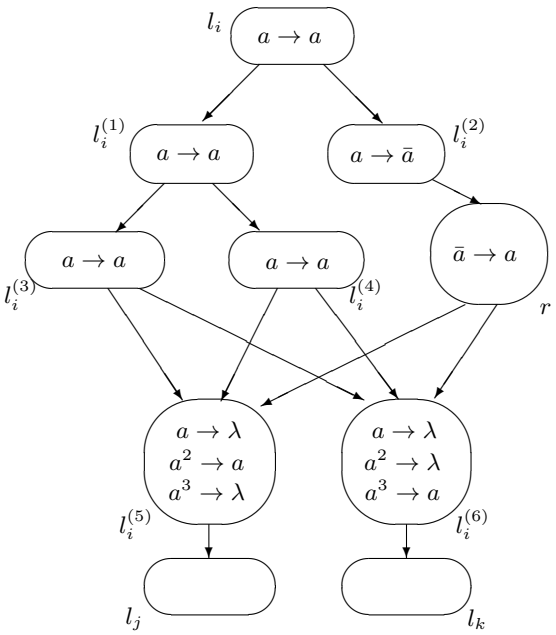


Fig. 1. Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

Figure 1 shows the module associated with an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$. The module is activated when neuron σ_{l_i} receives a spike. Initially, no neuron contains any spike, except for the neuron σ_{l_0} associated with l_0, the initial label of the register machine; each label has such an associated neuron, and also each register r has associated a neuron σ_r. Neuron σ_{l_i} sends a spike to neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. In the next step, neuron $\sigma_{l_i^{(2)}}$ sends an anti-spike to neuron σ_r, which corresponds to register r; at the same time, $\sigma_{l_i^{(1)}}$ sends a spike to the neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. If register r is non-empty, that is, neuron σ_r contains at least one a, then $\bar{a}$ removes one occurrence of a, which corresponds to subtracting one from register r, and no rule is applied in σ_r. This means that $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive only two spikes, from $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$, hence, σ_{l_j} is activated, whereas σ_{l_k} is not activated. If register r is empty, then the rule $\bar{a} \to a$ is used in σ_r, hence, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive three spikes, and this leads to the activation of σ_{l_k}, which is the correct continuation in this case.

The reader is referred to [17] for further details concerning the functioning of this module, and in general, for the proof of the universality of SN P systems with anti-spikes.

We cannot present all the developments concerning SN P systems with anti-spikes; most of the titles of the related articles listed at the end of the paper are self-explanatory. We only mention an important research direction in membrane computing in general and in the SN P systems area in particular, reminding the "old times" of investigations in formal language theory (see a survey in [7]) concerning the *descriptional complexity* of grammars and languages: considering size parameters for P systems. Because most of the classes of P systems are universal, for those classes the basic question is to find the smallest number of membranes in order to get the equivalence with Turing machines. For subuniversal classes, an important question of interest is whether or not the number of membranes induces an infinite hierarchy.

These questions are of interest for SN P systems, too, with or without anti-spikes. Further questions appear, resembling those mentioned in [7]: How many rules per neuron are needed? How many different types of neurons are needed? Can rules of a specific type be avoided?

Another question of interest is to find universal systems for a given class of devices with a small descriptional complexity; like in the case of universal Turing machines, we search for fixed P systems which can simulate any P system from a given class, as soon as the code of a particular system is introduced as an input to the universal one. For SN P systems, the "race" was started in [18], with several subsequent papers succeeding to decrease the complexity of the universal systems constructed there.

According to our knowledge, for SN P systems with anti-spikes the best results currently available are those from [24]: a universal system is constructed, for the case of computing functions, having 75 neurons and 125 rules, with 6 types of neurons and 8 types of rules. A related result is reported in [12], where a similar

system is described, containing 91 neurons, each of them containing only one rule, of one of the simple forms $a \to a$ and $a \to \bar{a}$. This once again proves the power of annihilation rules.

5 P Systems with Anti-Matter

The idea of considering "anti-matter" objects and their corresponding matter/anti-matter annihilation rules can be extended to all types of P systems. We briefly discuss it here for cell-like P systems.

Formally, a cell-like P system (of degree m, with catalysts) with anti-matter is a construct

$$\Pi = (O, A_O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{in}, i_{out})$$

where all the components are as in a usual P system and A_O is a set of symbols $\bar{a}$, for $a \in O \setminus C$ (obviously, we do not allow the catalysts to have anti-objects). In each compartment of μ we assume the matter/anti-matter annihilation rules $a\bar{a} \to \lambda$ to be present, for all $\bar{a} \in A_O$. As in SN P systems we might assume that these rules are used "automatically", in zero time, as soon as they can be applied. Yet in the following we assume the annihilation rules to be used as other rules, yet eventually with *weak priority* (e.g., see [2]) over all other rules, i.e., other rules then also may be applied if objects cannot be bound by some annihilation rule any more. In both cases, the rules in the sets R_i, $1 \leq i \leq m$, of the form $u \to v$ have to obey to the condition that neither u nor v may contain both the symbol a and its anti-matter object $\bar{a}$ for any $\bar{a} \in A_O$.

The functioning of such a system is as usual in membrane computing, keeping in mind that the annihilation rules have to be added to all sets of rules R_i, $1 \leq i \leq m$. By $NO_aP_m(ncoo, pri)$ we denote the family of sets of numbers generated by P systems with at most m membranes, using anti-objects, with non-cooperative rules. The parameter *pri* indicates the use of annihilation rules with priority over the other rules; it is omitted if we do not use this implicit priority. If in addition to non-cooperative rules we also allow catalytic rules and at most k catalysts, *ncoo* is replaced by cat_k in these notations.

Although the annihilation rules are expected to add a lot of computational power, it is still surprising that together with giving the annihilation rules priority over all other rules, non-cooperative rules are already sufficient to obtain computational completeness, whereas without this priority condition, in addition we need catalytic rules with one catalyst; in both cases rather simple proofs can be obtained, whereas without these matter/anti-matter annihilation rules, non-cooperative rules together with catalytic rules with two catalysts are needed, see the rather complex proof given in [5].

Theorem 1. $NO_aP_1(ncoo, pri) = NRE$.

Proof. Let $M = (3, H, l_0, l_h, I)$ (number of registers, labels of instructions, initial label, halt label, set if instructions) be a register machine with three registers; register 1 is the output register containing the result at the end of a successful computation, it is never decremented; registers 2 and 3 are empty at the begin and at the end of a successful computation. We now construct the (generating, hence, we omit i_{in}) P system with anti-matter

$$\Pi = (O, A_O, [\ ]_1, l_0, R_1, 1)$$

with only one membrane and the following components:

$$\begin{aligned} O &= \{l, l' \mid l \in H\} \cup \{a_r \mid r \in \{1,2,3\}\} \cup \{\#\}, \\ A_O &= \{\bar{a}_2, \bar{a}_3, \bar{\#}\}; \end{aligned}$$

the non-cooperative rules in R_1 are described below.

The contents of register r is represented by the number of copies of the object a_r, $r \in \{1,2,3\}$, in the system. The P system starts with the object l_0 representing the initial label of M.

For each instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in I, $r \in \{1,2,3\}$, we take the rules

$$\begin{aligned} &l_i \to l_j a_r \text{ and} \\ &l_i \to l_k a_r, \end{aligned}$$

which obviously simulate the given ADD-instruction.

For each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in I, $r \in \{2,3\}$, we consider the three rules

$$\begin{aligned} &l_i \to l_j \bar{a}_r, \\ &l_i \to l_i' \bar{a}_r, \\ &l_i' \to \# l_k. \end{aligned}$$

As rules common for all SUB-instructions, we also add the rules $\bar{a}_r \to \bar{\#}$, $r \in \{2,3\}$, the matter/antimatter annihilation rules $a_r \bar{a}_r \to \lambda$ and $\#\bar{\#} \to \lambda$ as well as the trap rules $\# \to \#\#$ and $\bar{\#} \to \#\#$.

When simulating a SUB-instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$, we have to make a non-deterministic choice between the decrement case and the zero-test. The decrement case of the SUB-instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated by the rule $l_i \to l_j \bar{a}_r$ and the subsequent application of the annihilation rule $a_r \bar{a}_r \to \lambda$. If this rule is not applicable, i.e., if register r is empty, the rule $\bar{a}_r \to \bar{\#}$ will be applied instead, which in absence of its counterpart $\#$ immediately evolves to $\#\#$ and thus leads to an infinite computation.

The zero-test is initiated with the rule $l_i \to l_i' \bar{a}_r$. If register r is empty, then $\bar{a}_r$ cannot be annihilated and therefore evolves to $\bar{\#}$, which then annihilates the symbol $\#$ generated by the rule $l_i' \to \# l_k$; if register r is not empty, $\bar{a}_r$ is annihilated by some copy of a_r, hence, the trap symbol $\#$ generated by the rule $l_i' \to \# l_k$ does not find its anti-matter $\bar{\#}$ and therefore evolves to $\#\#$, thus leading to an infinite computation. Here we find the crucial situation where we need the constraint that

annihilation rules have priority over all other rules, i.e., $\bar{a}_r \to \bar{\#}$ cannot be applied if the annihilation rule $a_r\bar{a}_r \to \lambda$ can be applied.

The rule $l_h \to \lambda$ is applied at the end of a successful simulation of the instructions of the register machine M, and the computation halts if no trap symbol # is present; the number of symbols a_1 in the skin membrane then represents the result of this halting computation. In sum, we obtain $N(M) = N(\Pi)$. □

Returning to descriptional complexity issues, it is worth noting that the P system constructed in the preceding proof has only one membrane and only three matter/anti-matter annihilation rules.

If we look for small universal systems, we may start with the universal register machine U_{32} from [9], with 8 registers which are decremented during the computations, and apply the construction given in the preceding proof, thus needing $8+1$ matter/anti-matter annihilation rules. An optimized P system with matter/anti-matter annihilation rules having priority over all other rules can be found in [1].

Without this priority of the annihilation rules, the construction is not working, hence, a characterization of the class $NO_aP_1(ncoo)$ remains as an open problem. Yet in addition using catalytic rules with one catalyst again allows us to obtain computational completeness:

Theorem 2. $NO_aP_1(cat_1) = NRE$.

Proof. We again consider a register machine $M = (3, H, l_0, l_h, I)$ as in the previous proof, and construct the (generating) catalytic P system

$$\Pi = (O, A_O, [\ ]_1, \{c\}, cl_0, R_1, 0)$$

with only one membrane (containing the single catalyst c) and the following components:

$$O = \{l, l', l'' \mid l \in H\} \cup \{a_r \mid r \in \{1,2,3\}\} \cup \{\#, d\},$$
$$A_O = \{\bar{a}_2, \bar{a}_3, \bar{\#}\};$$

the non-cooperative rules in R_1 are described below. The output symbols a_1 now are sent to the environment, in order not to have to count the catalyst in the skin membrane; for that purpose, we simply use the rule $a_1 \to (a_1, out)$.

For each instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in I, $r \in \{1,2,3\}$, we again take the rules

$$l_i \to l_j a_r \text{ and}$$
$$l_i \to l_k a_r.$$

For each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in I, $r \in \{2,3\}$, we now consider the following four rules:

$$l_i \to l_j \bar{a}_r,$$
$$l_i \to l_i'' d \bar{a}_r,$$
$$l_i'' \to l_i',$$
$$l_i' \to \# l_k.$$

As rules common for all SUB-instructions, we again add the matter/antimatter annihilation rules $a_r\bar{a}_r \to \lambda$ and $\#\bar{\#} \to \lambda$ as well as the trap rules $\# \to \#\#$ and $\bar{\#} \to \#\#$, but in addition, also $d \to \#\#$ as well as the catalytic rules $cd \to c$ and $c\bar{a}_r \to c\bar{\#}$, $r \in \{2, 3\}$.

The decrement case of the SUB-instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated as in the previous proof, by using the rule $l_i \to l_j\bar{a}_r$ and then applying the annihilation rule $a_r\bar{a}_r \to \lambda$. If this rule is not applicable, i.e., if register r is empty, the rule $\bar{a}_r \to \bar{\#}$ will be applied instead, which in absence of its counterpart $\#$ immediately evolves to $\#\#$ and thus leads to an infinite computation.

The zero-test now is initiated with the rule $l_i \to l_i''d\bar{a}_r$ thus introducing the (dummy) symbol d which keeps the catalyst busy for one step, where the catalytic rule $cd \to c$ has to be applied in order to avoid the application of the trap rule $d \to \#\#$. If register r is empty, then $\bar{a}_r$ cannot be annihilated and therefore evolves to $\bar{\#}$ in the third step by the application of the catalytic rule $c\bar{a}_r \to c\bar{\#}$, which symbol $\bar{\#}$ then annihilates the symbol $\#$ generated by the rule $l_i' \to \#l_k$ in the same step; if register r is not empty, $\bar{a}_r$ is annihilated by some copy of a_r already in the first step, hence, the trap symbol $\#$ generated by the rule $l_i' \to \#l_k$ does not find its anti-matter $\bar{\#}$ and therefore evolves to $\#\#$, thus leading to an infinite computation. Altough the annihilation rule $a_r\bar{a}_r \to \lambda$ now does not have priority over the catalytic rule $c\bar{a}_r \to c\bar{\#}$, maximal parallelism enforces $a_r\bar{a}_r \to \lambda$ to be applied, if possible, already in the first step instead of $c\bar{a}_r \to c\bar{\#}$, as in a successful derivation the catalyst c first has to eliminate the dummy symbol d.

The rule $l_h \to \lambda$ is applied at the end of a successful simulation of the instructions of the register machine M, and the computation halts if no trap symbol $\#$ is present; the number of symbols a_1 sent out to the environment during the computation represents the result of this halting computation. In sum, we obtain $N(M) = N(\Pi)$. □

6 Concluding Remarks

In this survey paper we have briefly recalled some basic ideas of membrane computing, and especially have given some information about spiking neural P systems, including spiking neural P systems with anti-spikes. We have also extended this idea of anti-objects ("anti-matter") to cell-like P systems with symbol objects, which can be proved to be computationally complete when the annihilation rules are applied with having priority over the remaining non-cooperative rules; without this priority, in addition catalytic rules with a single catalyst are needed to obtain computational completeness.

Several problems are still open in this area of P systems with anti-matter. Some of them have been formulated in this paper; the interested reader can find many more in the literature, for instance, in [6].

References

1. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and anti-matter in membrane systems. *Brainstorming Week in Membrane Computing*, Sevilla, February 2014.
2. A. Alhazov, D. Sburlan: Static Sorting P Systems. In: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.): Applications of Membrane Computing. *Natural Computing Series*, Springer, 2005, 215–252.
3. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*. Springer, Berlin, 2006.
4. P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer, Berlin, 2014.
5. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330 (2005), 251–266.
6. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Frontiers of membrane computing: Open problems and research topics, *Intern. J. Found. Computer Sci.*, 24, 5 (2013), 547–623.
7. J. Gruska: Descriptional complexity of context-free languages. *Proc. Symp. Math. Found. Computer Sci.*, High Tatras, 1973, 71–83.
8. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
9. I. Korec: Small universal Turing machines. *Theoretical Comp. Sci.*, 168 (1996), 267–301.
10. K. Krithivasan, V.P. Metta, D. Garg: On string languages generated by spiking neural P systems with anti-spikes. *Intern. J. Found. Computer Sci.*, 22, 1 (2011).
11. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.
12. V.P. Metta, A. Kelemenova: More on universality of spiking neural P systems with anti-spikes. Manuscript, 2013.
13. V.P. Metta, K. Krithivasan, D. Garg: Some characteristics of spiking neural P systems with anti-spikes. *Proc. 11th Intern. Conf. on Membrane Computing*, Jena, Germany, August 2010, 291–303.
14. V.P. Metta, K. Krithivasan, D. Garg: Modelling and analysis of spiking neural P systems with anti-spikes using Pnet lab. *Nano Comm. Networks*, 1, 2 (2011), 141–149.
15. V.P. Metta, K. Krithivasan, D. Garg: Computability of spiking neural P systems with anti-spikes. *New Math. and Natural Comput.*, 8, 3 (2012), 283–295.
16. V.P. Metta, K. Krithivasan, D. Garg: Spiking neural P systems with anti-spikes as transducers. *Romanian J. Info. Sci. and Tehnology*, 14, 1 (2011), 20–30.
17. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. *Int. J. Comoputers, Comm. and Control*, 4, 3 (2009), 273–282.
18. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90 (2007), 48–60.
19. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
20. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
21. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.

22. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 vols., Springer, Berlin, 1997.
23. T. Song, L. Pan, J. Wang, I. Venkat, K.G. Subramanian, R. Abdullah: Normal forms for spiking neural P systems with anti-spikes. *IEEE Trans. Nanobioscience*, 22, 4 (2012), 352–359.
24. T. Song, Y. Jiang, X. Shi, X. Zeng: Small universal spiking neural P systems with anti-spikes. *J. Comput. and Th. Nanoscience*, 10, 4 (2013), 999–1006.
25. T. Song, X. Wang, Z. Zhang, Z. Chen: Homogeneous spiking neural P systems with anti-spikes. *Neural Comput. and Applic.*, DOI 10.1007/s00521-0123-1397-8 (June 2013).
26. G. Tan, T. Song, Z. Chen, X. Zeng: Spiking neural P systems with anti-spikes and without annihilating priority working in a "flip-flop" way. *Intern. J. Computing Sci. and Math*, 4, 2 (2013), 152–162.
27. The P Systems Website: `www.ppage.psystems.eu`.

Probabilistic Guarded P Systems, A Formal Definition

Manuel García-Quismondo, Miguel A. Martínez-del-Amor,
Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
mgarciaquismondo@us.es, mdelamor@us.es, marper@us.es

Summary. In this paper, we extend the general framework of Multienvironment P systems, which is a formal framework for modelling the dynamics of population biology. The extension is made by a new variant within the probabilistic approach, called Probabilistic Guarded P systems (in short, PGP systems). We provide a formal definition, a simulation algorithm to capture the dynamics, and a survey of the associated software.

1 Introduction

Since P systems were introduced in 1998 [18], they have been utilised as a high level computational modelling framework [9, 19]. Their main advantage is the integration of the structural and dynamical aspects of complex systems in a comprehensive and relevant way, while providing the required formalisation to perform mathematical and computational analysis [2].

In this respect, multienvironment P systems are a general formal framework for population dynamics modelling in Biology [6]. This framework has two approaches: stochastic and probabilistic. Stochastic approach is usually applied to model *micro*-level systems (such as bacteria colonies), whereas the probabilistic approach is normally used for *macro*-level modelling (real ecosystems, for example). Population Dynamics P systems [2, 15, 16, 3] (PDP systems, in short) are a variant of multienvironment P systems, in the probabilistic approach. PDP systems have been successfully applied to ecological modelling, specially with real ecosystems of some endanger [5, 3] and exotic species [3]. PDP systems have shown to comply with four desirable properties of a computational model [2]: *relevance* (capture the essential features of the modelled system), *computability* (inherent by P systems), *understandability* (objects and rules capture the dynamics in a simple way), and *extensibility* (rule design is module-oriented).

In this paper, we introduce a brand new variant inside the probabilistic approach of multienvironment P systems: Probabilistic Guarded P systems (*PGP systems*, for short). They are specifically oriented for ecological processes. PGP systems are a computational probabilistic framework which takes inspiration from different Membrane Computing paradigms, mainly from Tissue–Like P systems [22], PDP systems [2] and Kernel P systems [11]. This framework aims for simplicity, considering these aspects:

Model designers: In PGP systems, model designers do not need to worry about context consistency. That is to say, they do not need to take into account that all rules simultaneously applied in a cell must define the same polarization in the right–hand side [15]. This is because the framework centralizes all context changes in a single rule per cycle, rather than distributing them across all rules. Therefore, there exist two types of rules: *context–changing* rules and *non context–changing* rules. Due to the nature of the model, only one of such rules can be applied at the same time on each cell, so context inconsistency is not possible. Moreover, the fact that the context is explicitly expressed in each cell and that cells do not contain internal cell structures simplifies transitions between contexts without loss of computational or modelling power.

Simulator developers: The fact that the framework implicitly takes care of context consistency simplifies the development of simulators for these models, as it is a non–functional requirement which does not need to be supported by simulators. In addition, the lack of internal structure in cells simplifies the simulation of object transmission; the model can be regarded as a set of memory regions with no hierarchical arrangement, thus enabling direct region fetching.

Probabilistic Guarded P Systems can be regarded as an evolution of Population Dynamic P systems. In this context, PGP systems propose a modelling framework for ecology in which inconsistency (that is to say, undefined context of membranes) is handled by the framework itself, rather than delegating to simulation algorithms. In addition, by replacing alien concepts to biology (such as electrical polarizations and internal compartment hierarchies) by state variables known as *flags* and defined by designers models are more natural to experts, thus simplifying communication between expert and designer.

This paper is structured as follows. Section 2 introduces some preliminaries. Section 3 shows the formal framework of multienvironment P systems, and the two main approaches. Section 4 describes the framework of PGP systems, providing a formal definition, some remarks about the semantics of the model, and a comparison with other similar frameworks of Membrane Computing. Section 5 provides a simulation algorithm, and a software environment based on P–Lingua and a C++ simulator. Section 6 summarizes an ecosystem under study with PGP systems. Finally, Section 7 ends the paper with conclusions and future work.

2 Preliminaries

An *alphabet* Γ is a non–empty set whose elements are called *symbols*. An ordered finite sequence of symbols of Γ is a *string* or *word* over Γ. As usual, the empty string (with length 0) will be denoted by λ. The set of all strings over an alphabet Γ is denoted by Γ^*. A *language* over Γ is a subset of Γ^*.

A *multiset* m over an alphabet Γ is a pair $m = (\Gamma, f)$ where $f : \Gamma \to \mathbb{N}$ is a mapping. For each $x \in \Gamma$ we say that $f(x)$ is the *multiplicity* of the symbol x in m. If $m = (\Gamma, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in \Gamma \mid f(x) > 0\}$. A multiset is finite if its support is a finite set. A *set* is a multiset such that the multiplicity of each element of its support, is equal to 1.

If $m = (\Gamma, f)$ is a finite multiset over Γ, and $supp(m) = \{a_1, \ldots, a_k\}$ then it will be denoted as $m = a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ (here the order is irrelevant), and we say that $f(a_1) + \cdots + f(a_k)$ is the cardinal of m, denoted by $|m|$. The empty multiset is denoted by $\emptyset$. We also denote by $M_f(\Gamma)$ the set of all finite multisets over Γ.

Let $m_1 = (\Gamma, f_1)$ and $m_2 = (\Gamma, f_2)$ multisets over Γ. We define the following concepts:

- The union of m_1 and m_2, denoted by $m_1 + m_2$ is the multiset (Γ, g), where $g = f_1 + f_2$, that is, $g(x) = f_1(x) + f_2(x)$ for each $x \in \Gamma$.
- The relative complement of m_2 in m_1, denoted by $m_1 \setminus m_2$ is the multiset (Γ, g), where $g = f_1(x) - f_2(x)$ if $f_1(x) \geq f_2(x)$ and $g(x) = 0$ otherwise.

We also say that m_1 is a submultiset of m_2, denoted by $m_1 \subseteq m_2$, if $f_1(x) \leq f_2(x)$ for each $x \in \Gamma$.

Let $m = (\Gamma, f)$ a multiset over Γ and A a set. We define the intersection $m \cap A$ as the multiset (Γ, g), where $g(x) = f(x)$ for each $x \in \Gamma \cap A$, and $g(x) = 0$ otherwise.

3 Multienvironment P systems

Definition 1. *A multienvironment P system of degree (q, m, n) with $q \geq 1$, $m \geq 1$, taking T time units, $T \geq 1$, is a tuple*

$$\Pi = (G, \Gamma, \Sigma, \Phi, T, n, \{\Pi_{k,j} \mid 1 \leq k \leq n,\ 1 \leq j \leq m\}, \{(f_j, E_j) \mid 1 \leq j \leq m\}, \mathcal{R}_E)$$

where:

- *$G = (V, S)$ is a directed graph. Let $V = \{e_1, \ldots, e_m\}$ whose elements are called environments;*
- *Γ, Σ and Φ are finite alphabets such that $\Sigma \subsetneq \Gamma$ and $\Gamma \cap \Phi = \emptyset$.*
- *T, n are natural numbers*
- *For each k, j ($1 \leq k \leq n$, $1 \leq j \leq m$), $\Pi_{k,j}$ is a tuple $(\Gamma, \mu, \mathcal{M}_{1,j}^k, \ldots, \mathcal{M}_{q,j}^k, \mathcal{R}_j, i_{in})$, where:*

  - *μ is a rooted tree with $q \geq 1$ nodes labelled by elements from $\{1, \ldots, q\} \times \{0, +, -\}$.*
  - *For each i, $1 \leq i \leq q$, $\mathcal{M}_{i,j}^k \in M_f(\Gamma)$.*
  - *$\mathcal{R}_j$ is a finite set of rules of the type: $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$, being $u, v, u', v' \in M_f(\Gamma)$, $1 \leq i \leq q$, $\alpha, \alpha' \in \{0, +, -\}$ and p is a computable function whose domain is $\{0, \ldots, T\}$.*
  - *i_{in} is a node from μ.*
- *For each j, $1 \leq j \leq m$, $f_j \in \Phi$ and $E_j \in M_f(\Sigma)$.*
- *$\mathcal{R}_E$ is a finite set of rules among environments of the types:*

$$(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}} \qquad (\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j_1}}$$
$$\{f\}\,(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}} \qquad \{f\}\,(u, f)_{e_j} \xrightarrow{p_4} (v, g)_{e_j}$$

being $x, y_1, \ldots y_h \in \Sigma$, $(e_j, e_{j_i}) \in S$, $1 \leq j \leq m, 1 \leq i \leq h$, $1 \leq k \leq n, f, g \in \Phi, u, v \in M_f(\Gamma)$ and p_1, p_2, p_3, p_4 are computable functions whose domain is $\{0, \ldots, T\}$.

In other words, a system as described in the previous definition can be viewed as a set of m environments $e_1, \ldots, e_m$ linked between them by the arcs from the directed graph G. Each environment e_j has a flag from Φ at any instant and also it can contains objects from Σ and P systems of the type $\Pi_{k,j} = (\Gamma, \mu, \mathcal{M}_{1,j}^k, \ldots, \mathcal{M}_{q,j}^k, \mathcal{R}_j^k, i_{in})$. Multisets $\mathcal{M}_{1j}^k, \ldots, \mathcal{M}_{q,j}^k$ describe the initial multisets of $\Pi_{k,j}$ corresponding to this environment. Every rule $r \in \mathcal{R}_j^k$ has a computable function $f_{r,j}$ (specific for environment j) associated with it.

In total, there are n systems $\Pi_{k,j}$, all of them with the same skeleton (identical working alphabets, objects and **flags**, the same membrane structure and the same rules $u[v]_i^\alpha \longrightarrow u'[v']_i^{\alpha'}$, specified in each environment (independently of k) through the computable function $f_{r,j}$ associated with them).

A *configuration* of the system at any instant t is a tuple whose components are the following: (a) the flags associated with each environment at instant t (initially $f_1, \ldots, f_m$); (b) the multisets of objects present in the m environments at instant t (initially $E_1, \ldots, E_m$); and (c) the multisets of objects associated with each of the regions of each P system $\Pi_{k,j}$ (initially $\mathcal{M}_{1,j}^k, \ldots, \mathcal{M}_{q,j}^k$), together with the polarizations of their membranes (initially all membranes have a neutral polarization).

We assume that a global clock exists, marking the time for the whole system, that is, all membranes and the application of all rules (both from $\mathcal{R}_\mathcal{E}$ and $\mathcal{R}$) are synchronized in all environments.

The P system can pass from one configuration to another by using the rules from $\mathcal{R} = \mathcal{R}_E \cup \bigcup_{j=1}^m \mathcal{R}_j^k$ as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, and all applicable rules are simultaneously applied.

A rule of the type $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration membrane i

has polarization α, contains multiset v and its parent (the environment if the membrane is the skin membrane) contains multiset u. When that rule is applied, multisets u, v produce u', v', respectively, and the new polarization is α' (the value of function p in that moment provide the affinity of the application of that rule). For each j $(1 \leq j \leq m)$ there is just one further restriction, concerning the consistency of charges: in order to apply several rules of $\mathcal{R}_j^k$ simultaneously to the same membrane, all the rules must have the same electrical charge on their right-hand side.

A rule of the environment of the type $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j contains object x. When that rule is applied, object x passes from e_j to $e_{j_1}, \ldots, e_{j_h}$ possibly transformed into objects $y_1, \ldots, y_h$, respectively (the value of function p_1 in that moment provide the affinity of the application of that rule).

A rule of the environment of the type $(\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j'}}$: is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j contains the P system $\Pi_{k,j}$. When that rule is applied, the system $\Pi_{k,j}$ passes from environment e_j to environment $e_{j'}$ (the value of function p_2 in that moment provide the affinity of the application of that rule).

A rule of the environment of the type $\{f\}(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j has flag f and contains the multiset u. When that rule is applied multiset u produces multiset v and environment e_j keep the same flag. This kind of rule can be applied many times in a computation step. The value of function p_3 in that moment provide the affinity of the application of that rule.

A rule of the environment of the type $\{f\}(u, f)_{e_j} \xrightarrow{p_4} (v, g)_{e_j}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j has flag f and contains the multiset u. When that rule is applied multiset u produces multiset v and flag f of environment e_j is replaced by flag g. Bearing in mind that each environment only has a flag in any instant, this kind of rules can only be applied once in any moment. Hence, the value of the function p_4 in any instant is equal to 1.

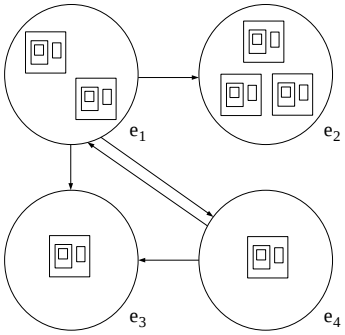
Next, we depict the two approaches (stochastic and probabilistic) for multienvironment P systems.

3.1 Stochastic approach

We say that a multienvironment P system has a stochastic approach if the following holds:

(a) The alphabet of flags, Φ, is an empty set.
(b) The computable functions associated with the rules of the P systems are **propensities** (obtained from the kinetic constants): These rules is function of the time but they do not depend on the environment.

(c) The P systems $\Pi_{k,j}$ do not depend on index j, this index is irrelevant in this approach.
(d) Initially, the P systems $\Pi_{k,j}$ are randomly distributed among the m en environments of the system.



Multicompartmental P systems

Multicompartmental P systems are multienvironment P systems with a stochastic approach which can be formally expressed as follows:

$$\Pi = (G, \Gamma, \Sigma, T, n, \{\Pi_{k,j} \,|\, 1 \leq k \leq n,\, 1 \leq j \leq m\}, \{E_j \,|\, 1 \leq j \leq m\}, \mathcal{R}_E)$$

These systems can be viewed as a set of m environment connected by the arcs of a directed graph G. Each environment e_j only can contains P systems of the type $\Pi_{k,j}$. The total number of P systems is n, all of them with the same skeleton. The functions associated with the rules of the system are propensities which are computed as follows: stochastic constants are computed from kinetic constants by applying the mass action law, and the propensities are obtained from the stochastic constants by using the concentration of the objects in the LHS at any instant. In these systems there are rules of the following types:

1. $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$
2. $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$
3. $(\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j'}}$

The dynamics of these systems is captured by the multicompartmental Gillespie's algorithm [21] or the deterministic waiting time [4]. A software environment supporting this model is Infobiotics Workbench [1], which provides (in version 0.0.1): a modelling language, a multi-compartmental stochastic simulator based on Gillespies Stochastic Simulation Algorithm, a formal model analysis, and a structural and parameter model optimisation.

3.2 Probabilistic approach

We say that a multienvironment P system has a stochastic approach if the following holds:

(a) The total number of P systems $\Pi_{k,j}$ is, at most, the number m of environment, that is, $n \leq m$.

(b) Functions p_r associated with rule $r \equiv u[v]_i^\alpha \xrightarrow{p_r} u'[v']_i^{\alpha'}$ from $\Pi_{k,j}$ are **probability functions** such that for each $u, v \in M_f(\Gamma)$, $i \in \{1, \ldots, q\}$, $\alpha \in \{0, +, -\}$, if $r_1, \ldots, r_z$ are the rules in R_j^k whose LHS is $u\ [\ v\ ]_i^\alpha$, then $\sum_{j=1}^{z} p_{r_j}(t) = 1$, for each t $(1 \leq t \leq T)$.

(c) Functions p_1 associated with the rules of the environment $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ are **probability functions** such that for each $x \in \Sigma$ and each environment e_j, the sum of all functions associated with the rules whose LHS is $(x)_{e_j}$, is equal to 1.

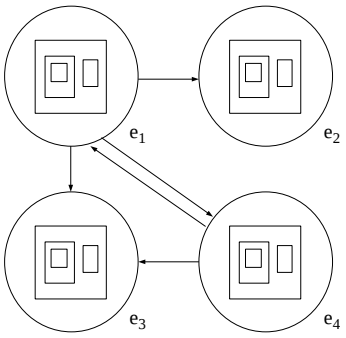
(d) Functions p_2 associated with the rules of the environment $(\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j'}}$ are constant functions equal to 0; that is, these rules will never be applied.

(e) Functions p_3 associated with the rules of the environment $\{f\}\,(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}}$ are **probability functions**.

(f) Functions p_4 associated with the rules of the environment $\{f\}\,(u, f)_{e_j} \xrightarrow{p_4} (v, g)_{e_j}$ are constant functions equal to 1.

(g) There is no rules $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$ in the skin membranes of $\Pi_{k,j}$ and rules of the environment $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ such that $x \in u$.

(h) Initially, each environment e_j contains at most one P system $\Pi_{k,j}$.



Population Dynamics P systems (PDP)

Population Dynamics P systems are multienvironment P systems with a probabilistic approach such that the alphabet Φ of the flags is an empty set and $n = m$, that is, the environment have not any flag and the total number n of P systems are equal to the number m of environments. Then in a PDP system each environment e_j contains exactly one P system $\Pi_{k,j}$ which will be denoted by Π_j

$$\Pi = (G, \Gamma, \Sigma, T, n, \{\Pi_j \mid 1 \leq j \leq m\}, \{E_j \mid 1 \leq j \leq m\}, \mathcal{R}_E)$$

In these systems there are rules of the following types:

1. $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$
2. $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$

Let us recall that in theses kind of systems each rule has associated a probability function that depends on the time and on the environment where the rule is applied.

The dynamics of these systems is captured by the **D**irect **N**on-deterministic **D**istribution algorithm with **P**robabilities (**DNDP**) algorithm [16], or the **D**irect distribution based on **C**onsistent **B**locks **A**lgorithm (**DCBA**) [15]. DNDP aims to perform a random distribution of rule applications without using the concept of rule block, but this selection process is biased towards those rules with the highest probabilities. DCBA was first conceived to overcome the accuracy problem of DNDP, by performing an object distribution along the rule blocks, before applying the random distribution process. Although the accuracy achieved by the DCBA is better than the DNDP algorithm, the latter is much faster. In order to improve the performance of simulators implementing DCBA, parallel architectures has been used [14]. For example, a GPU-based simulator, using CUDA, reaches the acceleration of up to 7x, running on a NVIDIA Tesla C1060 GPU (240 processing cores). However, these accelerated simulators are still to be connected to those general environments to run virtual experiments. Therefore, P–Lingua and pLinguaCore are being utilised to simulate PDP systems [2, 10]. The provided virtual experimentation environment is called MeCoSim [20], and it is based on P–Lingua.

4 Probabilistic Guarded P systems (PGP)

Probabilistic Guarded P systems are multienvironment P systems with a probabilistic approach such that $n = 0$, that is, there is no P systems $\Pi_{k,j}$ (so the alphabet Γ can be considered as an emptyset), and the alphabet of the environment, Σ, and the alphabet of the flags, Φ are disjoint.

Definition 2. *A Probabilistic Guarded P system* (PGP system*, for short*) *of degree* $m \geq 1$ *is a tuple* $\Pi = (G, \Sigma, \Phi, T, \{(f_j, E_j) \mid 1 \leq j \leq m\}, \mathcal{R}_E)$*, where:*

- $G = (V, S)$ *is a directed graph whose set of nodes is* $V = \{e_1, \ldots, e_m\}$.
- Σ *and* Φ *are finite alphabets such that* $\Sigma \cap \Phi = \emptyset$. *Elements in* Σ *are called* ***objects*** *and elements in* Φ *are called* ***flags***.
- $\mathcal{R}_E$ *is a finite set of rules of the following types:*
  - $\{f\}(u)_{e_j} \to (v)_{e_{j_1}}$ *with* $u, v \in M_f(\Sigma)$, $f \in \Phi$ *and* $1 \leq j, j_1 \leq m$.
  - $\{f\}(u, f)_{e_j} \to (v, g)_{e_j}$ *with* $u, v \in M_f(\Sigma)$, $f, g \in \Phi$ *and* $1 \leq j \leq m$.

  There is no rules of types $\{f\}(u, f)_{e_j} \to (v, g)_{e_j}$ *and* $\{f\}(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}}$, *for* $f \in \Phi$, $1 \leq j, j_1 \leq m$ *and* $u \in M_f(\Sigma)$.

  For each $f \in \Phi$ *and* $j, 1 \leq j \leq m$, *there exists only one rule of type* $\{f\}(u, f)_{e_j} \to (v, g)_{e_j}$.
- *The arcs of graph* $G = (V, S)$ *is defined from* $\mathcal{R}_E$ *as follows:* $(e_j, e_{j_1}) \in S$ *if and only if there exists a rule of the type* $\{f\}(u)_{e_j} \to (v)_{e_{j_1}}$, *or* $j = j_1$ *and there exists a rule of the type* $\{f\}(u, f)_{e_j} \to (v, g)_{e_j}$.
- *Each rule from* $\mathcal{R}_E$ *has associated a probability, that is, there exists a function* $p_{\mathcal{R}_E}$ *from* $\mathcal{R}_E$ *into* $[0, 1]$, *such that:*
  - *For each* $f \in \Phi, u \in M(\Sigma), 1 \leq j \leq m$, *if* $r_1, \ldots, r_t$ *are rules of the type* $\{f\}(u)_{e_j} \to (v)_{e_{j_1}}$, *then* $\sum_{s=1}^{t} p_{\mathcal{R}_E}(r_s) = 1$.
  - *If* $r \equiv \{f\}(u, f)_{e_j} \to (u, g)_{e_j}$, *then* $p_{\mathcal{R}_E}(r) = 1$.
- *For each* $j, 1 \leq j \leq m$, *we have* $f_j \in \Phi$ *and* $E_j \in M_f(\Sigma)$.

A Probabilistic Guarded P system can be viewed as a set of m environments, called *cells*, labelled by $1, \ldots, m$ such that: (a) $E_1, \ldots, E_m$ are finite multisets over Σ representing the objects initially placed in the cells of the system; (b) $f_1, \ldots, f_m$ are flags that initially mark the cells; (c) G is a directed graph whose arcs specify connections among cells; (d) $\mathcal{R}_E$ is the set of rules that allow the evolution of the system and each rule r is associated with a real number $p_{\mathcal{R}_E}(r)$ in $[0, 1]$ describing the probability of that rule to be applied in the case that it is applicable.

In PGP systems, two types of symbols are used: *objects* (elements in Σ) and *flags* (elements in Φ). It can be considered that objects are **in** cells and flags are **on** (the borderline of) cells.

A *configuration* of a PGP system at any instant t is a tuple whose components are the following: (a) the flags associated with each cell at instant t (initially $f_1, \ldots, f_m$), and (b) the multisets of objects present in the m cells at instant t (initially $E_1, \ldots, E_m$).

Finally, in order to ease the understandability of the whole framework, Figure 1 shows a graphical summary of multienvironment P systems and the two approaches (stochastic and probabilistic).

4.1 Semantics of PGP systems

Definition 3. *A* configuration *at any instant* $t \geq 0$ *of a PGP system* Π *is a tuple* $\mathcal{C}_t = (x_1, u_1, \ldots, x_m, u_m)$ *where, for each* i, $1 \leq i \leq m$, $x_i \in \Phi$ *and* $u_i \in M(\Sigma)$. *That is to say, a configuration of* Π *at any instant* $t \geq 0$ *is described by all multisets of objects over* Σ *associated with all the cells present in the system and the flags*

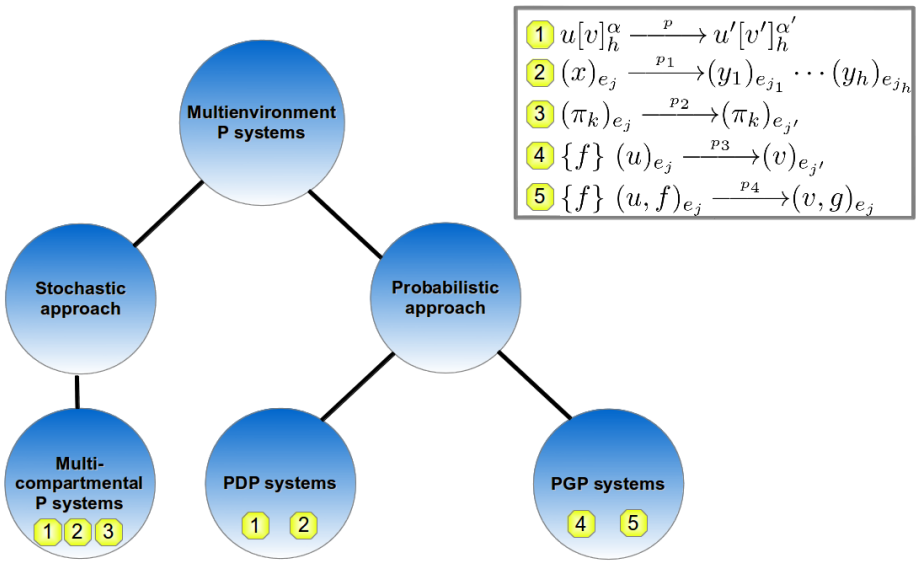


Fig. 1: The formal framework of Multienvironment P systems

marking these cells. $(f_1, E_1, \ldots, f_m, E_m)$ *is said to be the* initial configuration *of* Π*. At any instant, each cell has one and only one flag, in a similar manner to polarizations in cell–like P systems.*

Definition 4. *A rule* r *of the type* $\{f\}\,(u)_i \to (v)_j$ *is* applicable *to a configuration* $\mathcal{C}_t = (x_1, u_1, \ldots x_m, u_m)$ *if and only if* $x_i = f$ *and* $u \subseteq u_i$*, for all* $1 \leq i \leq m$*.*

When applying r *to* C_t*, objects in* u *are removed from cell* i *and objects in* v *are produced in cell* j*. Flag* f *is not changed; it plays the role of a catalyst assisting the evolution of objects in* u*.*

Definition 5. *A rule* r *of the type* $\{f\}\,(u,f)_i \to (v,g)_i$ *is* applicable *to a configuration* $\mathcal{C}_t = (x_1, u_1, \ldots x_m, u_m)$ *if and only if* $x_i = f$ *and* $u \subseteq u_i$*, for all* $1 \leq i \leq m$*.*

When applying r *to* C_t*, in cell* i *objects in* u *are replaced by those in* v *and* f *is replaced by* g*. In this case, Flag* f *is consumed, so* r *can be applied only once in instant* t *in cell* i*.*

Remark 1. After applying a rule r of the type $\{f\}\,(u,f)_i \to (v,g)_i$, other rules r' of the type $\{f\}\,(u)_i \to (v)_j$ can still be applied (the flag remains in vigour). However, f has been consumed, so no more rules of the type $\{f\}\,(u,f)_i \to (v,g)_i$ can be applied.

Definition 6. *A configuration is a* halting configuration *if no rule is applicable to it.*

Definition 7. *We say that configuration* $\mathcal{C}_1$ yields *configuration* $\mathcal{C}_2$ *in a transition step if we can pass from* $\mathcal{C}_1$ *to* $\mathcal{C}_2$ *by applying rules from* $\mathcal{R}_\mathcal{E}$ *in a non–deterministic, maximally parallel manner, according to their associated probabilities denoted by map* $p_{\mathcal{R}_\mathcal{E}}$*. That is to say, a maximal multiset of rules from* $\mathcal{R}_\mathcal{E}$ *is applied, no further rule can be added.*

Definition 8. *A* computation *of a PGP system* Π *is a sequence of configurations such that: (a) the first term of the sequence is the initial configuration of* Π*, (b) each remaining term in the sequence is obtained from the previous one by applying the rules of the system following Definition 7, (c) if the sequence is finite (called halting computation) then the last term of the system is a halting configuration.*

4.2 Comparison between PGP systems and other frameworks in Membrane Computing

Probabilistic Guarded P systems (*PGP systems*) display similarities with other frameworks in Membrane Computing. As a sample, in *P systems with proteins on membranes* are a type of cell-like systems in which membranes might have attached a set of proteins which regulate the application of rules, whilst in PGP systems each cell has only one flag. Therefore, some rules are applicable if and only if the corresponding protein is present. More information about this kind of P systems can be found in [17].

When comparing PGP systems and *Population Dynamics P systems* [2], it is important to remark the semantic similarity between flags and polarizations, as they both define at some point the context of each compartment. Nevertheless, as described at the beginning of this chapter, upon the application of a rule $r \equiv \{f\}\,(u, f)_i \rightarrow (v, g)_i$ flag f is consumed, thus ensuring that r can be applied at most once to any configuration. This property keeps PGP transitions from yielding inconsistent flags; at any instant, only one rule at most can change the flag in each membrane, so scenarios in which inconsistent flags produced by multiple rules are impossible. Moreover, in PDP systems the number of polarizations is limited to three (+, - and 0), whereas in their PGP counterpart depends on the system itself. Finally, each compartment in PDP systems contains a hierarchical structure of membranes, which is absent in PGP systems. Figure 2 summarizes this comparison.

5 Simulation of PGP systems

When simulating PGP systems, there exist two cases, according to if there exists object competition or not. In this work, only algorithms for the second case are

	PGP systems	**P systems with proteins**	**PDP systems**
Structure	Tissue–like (given by a directed graph)	Cell–like (given by a rooted tree)	Tissue–like (given by a directed graph of environments containing a rooted tree each)
Rule	Each left–hand side contains one flag and a multiset of objects	Each left–hand side contains one protein and one object	Each left–hand side contains one polarization and a multiset of objects
Affected compartments	The application of a rule might affect, at most, two cells in the system	The application of a rule affects one and only one cell in the system	The application of a rule might affect, at most, two cells in the system
Number of applications	Each rule of type $r \equiv \{f\}\,(u, f)_i \to (v, g)_i$ can be applied, at most, only once to any configuration	Every rule is possible to be applied multiple times to any configuration	Every rule is possible to be applied multiple times to any configuration
Number of flags	For each configuration, there exists only one flag per cell	For each configuration, there might exist multiple proteins per cell	For each configuration, there exists only one polarization per cell

Fig. 2: Comparison of PGP systems, PDP systems and P systems with proteins

introduced, but some ideas are given to handle object competition among rules in the model, and kept for future developments.

5.1 Some definitions on the model

As it is the case in Population Dynamic P systems, in PGP systems some definitions are introduced prior to describing simulation algorithms. It must be noted that these concepts are analogous to those described in [15], but obviously adapted to the syntax of PGP systems.

Remark 2. For the sake of simplicity, henceforth the following notation will be used. For every cell i , $1 \leq i \leq m$, and time t , $0 \leq t \leq T$, the flag and multiset of cell i in step t are denoted as $x_{i,t} \in \Phi$ and $u_{i,t} \in M(\Sigma)$, respectively. Similarly, $u(y)$, where $u \in M(\Sigma),\ y \in \Sigma$ denote the number of objects y in multiset u.

Definition 9 shows the notation regarding the left-hand and right-hand sides of rules.

Definition 9. *For each rule $r \in \mathcal{R}_{\mathcal{E}}$:*

Type 1: If r is of the form $r \equiv \{f\}\,(u)_i \rightarrow (v)_j$, we denote the left–hand side as $LHS(r) = (i, f, u)$ and the right–hand side as $RHS(r) = (j, f, v)$.
Type 2: If r is of the form $r \equiv \{f\}\,(u, f)_i \rightarrow (v, g)_i$, we denote the left–hand side as $LHS(r) = (i, f, u, f)$ and the right–hand side as $RHS(r) = (i, g, v)$.

Let us recall that for each i , $1 \leq i \leq m$ and $f \in \Phi$, there exists an *unique* rule of type 2: $r \equiv \{f\}\,(u, f)_i \rightarrow (v, g)_i$.

Next, Definition 10 introduces the concept of rule blocks in PGP systems, which is inspired by the one used in PDP systems [15].

Definition 10. *For each $1 \leq i \leq q$, $f \in \Phi$, and $u \in M(\Sigma)$, we will denote:*

- *The* block of communication rules $B^1_{i,f,u} = \{r \in \mathcal{R} : LHS(r) = (i, f, u)\}$*; that is, the set of rules of type 1 having the same left–hand side.*
- *The* block of context–changing rules $B^2_{i,f,u} = \{r \in \mathcal{R} : LHS(r) = (i, f, u, f)\}$*; that is, the set of rules of type 2 having the same left–hand side.*

Obviously, $B^1_{i,f,u} \cap B^2_{i,f,u} = \emptyset$. It is important to recall that, as it is the case in PDP systems, the sum of probabilities of all the rules belonging to the same block is always equal to 1 – in particular, rules with probability equal to 1 form individual blocks. Consequently, blocks of context–changing rules (type 2) are composed solely of a rule. In addition, rules with overlapping (but different) left–hand sides are classified into different blocks.

Definition 11. *For each $i, 1 \leq i \leq m$, we will consider the set of all rule blocks associated with cell i as $B_i = \{B^1_{i,f,u}, B^2_{i,f,u} : f \in \Phi \wedge u \in M(\Sigma)\}$.*

We will also consider a total order in B_i, for $1 \leq i \leq m$, $B_i = \{B_{i,1}, B_{i,2}, \ldots, B_{i,\alpha_i}\}$. Therefore, there are α_i blocks associated to cell i.

Furthermore, let $B_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq \alpha_i$ be a block associated to cell i. We define the following notations:

- $Type(B_{i,j})$ is equal to:
  - 1, if $\exists f \in \Phi, u \in M(\Sigma)$ such that $B_{i,j} = B^1_{i,f,u}$
  - 2, if $\exists f \in \Phi, u \in M(\Sigma)$ such that $B_{i,j} = B^2_{i,f,u}$
- $Flag(B_{i,j}) = f$, if $\exists k(1 \leq k \leq 2) \wedge \exists u \in M(\Sigma)$ such that $B_{i,j} = B^k_{i,f,u}$
- $Mult(B_{i,j}) = u$, if $\exists k(1 \leq k \leq 2) \wedge \exists f \in \Phi$ such that $B_{i,j} = B^k_{i,f,u}$

In addition, for each block $B_{i,j}$, $1 \leq i \leq m$ and $1 \leq j \leq \alpha_i$, associated to cell i, we consider a total order in the set of integrated rules: $B_{i,j} = \{r_{i,j,1}, \ldots, r_{i,j,h_{i,j}}\}$, where $h_{i,j}(1 \leq i \leq m,\ 1 \leq j \leq \alpha_i)$ denotes the number of rules in block $B_{i,j}$. Obviously, all the rules of a block are of the same type.

Definition 12. *A PGP system is said to feature* object competition*, if there exists at least two different blocks $B_{i,j}$ and $B_{i,j'}$ (possibly of different type), such that $Flag(B_{i,j}) = Flag(B_{i,j'})$, and $Mult(B_{i,j}) \cap Mult(B_{i,j'}) \neq \emptyset$. That is, their rules have overlapping (but not equal) left-hand sides.*

Remark 3. It is worth noting that all rules in the model can be consistently applied. This is because there can only exists one flag $f \in \Phi$ at every membrane at the same time, and, consequently, at most one context–changing rule $r \equiv \{f\}\,(u, f)_i \to (v, g)_i$ can consume f and replace it (where possibly $f = g$).

Definition 13. *Given a block $B^1_{i,f,u}$ or $B^2_{i,f,u}$, where $u \in M(\Sigma), f \in \Phi$, $1 \leq i \leq m$ and a configuration $C_t = \{x_1, u_1, \ldots, x_m, u_m\}$, $0 \leq t \leq T$, the* maximum number of applications *of such a block in C_t is the maximum applications of any of its rule in C_t.*

5.2 Simulation Algorithm

Next, we define some auxiliary data structures to be used in the simulation algorithms.

NBA (Number of Block Applications): a matrix of integer numbers of dimension $m \times N_{BM}$, where $N_{BM} = max(\alpha_i),\ 1 \leq i \leq m$ (maximum number of blocks for all cells). Each element $NBA_{i,j},\ 1 \leq i \leq m,\ 1 \leq j \leq N_{BM}$ stores the number of applications of block $B_{i,j}$.

NRA (Number of Rule Applications): a matrix of integer numbers of dimension $m \times N_{BM} \times N_{RM}$, where $N_{RM} = max(h_{i,j}),\ 1 \leq i \leq m,\ 1 \leq j \leq \alpha_i$ (maximum number of rules for all blocks in all membranes). Each element $NRA_{i,j,k},\ 1 \leq i \leq m,\ 1 \leq j \leq \alpha_i,\ 1 \leq k \leq h_{i,j}$, stores the number of applications of rule $r_{i,j,k}$, identified by its cell, block and local identifier inside its block, according to the established total order.

The algorithm for simulation of PGP systems receives three parameters:

- The PGP system Π of degree m.
- The integer number $T > 0$ (number of time steps).
- An integer number $K > 0$ (random accuracy). It indicates for how many cycles block applications are assigned among their rules in random fashion. That is, the algorithm distributes the applications of each block among its rules for K cycles, and after that, block applications are maximally assigned among rules in a single cycle. It is used as an accuracy parameter for the probabilistic method. Algorithm 5.4 performs this function.

When simulating PGP systems without object competition, it is not necessary to randomly assign objects among blocks; as they do not compete for objects, then the number of times that each block is applied is always equal to its maximum number of applications. As it is the case of DCBA for PDP systems [15], the simulation algorithm heavily relies on the concept of block, being rule applications secondary. However, DCBA handles object competition among blocks, penalizing more those blocks which require a larger number of copies of the same object which is inspired by the amount of energy required to join individuals from the same species. On the other hand, object competition is not supported on the proposed algorithm. Algorithm 5.1 describes a simulation algorithm for PGP systems without object competition.

Algorithm 5.1 Algorithm for simulation of PGP systems

Input:

- T: an integer number $T \geq 1$ representing the iterations of the simulation.
- K: an integer number $K \geq 1$ representing non–maximal rule iterations (i.e., iterations in which the applications selected for each rule do not necessarily need to be maximal).
- $\Pi = (G, \Sigma, \Phi, T, \{(f_j, E_j) \mid 1 \leq j \leq m\}, \mathcal{R}_E)$: a PGP system of degree $m \geq 1$.

```
Initialization (Π)
for t ← 1 to T do                                          ▷ See Algorithm 5.2
    C'_t ← C_{t−1}
    SELECTION of rules:
        PHASE 1: Objects distribution (C'_t)               ▷ See Algorithm 5.3
        PHASE 2: Rule application distribution (C'_t)      ▷ See Algorithm 5.4
    EXECUTION of rules:
        PHASE 3: Object production (C'_t)                  ▷ See Algorithm 5.5
    C_t ← C'_t
end for
```

On each simulation step t, $1 \leq t \leq T$ and cell i, $1 \leq i \leq m$, the following stages are applied: *Object distribution* (selection), *Rule application distribution* (selection) and *Object generation* (execution).

However, before starting the simulation process, we must initialize some data structures. In *Initialization* (Algorithm 5.2), the initial configuration C_0 is constructed with the input PGP system Π. Moreover, the information about blocks are created; that is, the blocks of rules are computed, and ordered for each cell. Moreover, the rules inside each block are also ordered. Finally, the data structures NBA and NRA are initialized with zeros.

In the *Object distribution* stage (Algorithm 5.3), objects are distributed among blocks. As the system to simulate does not feature object competition, the number of applications of each block is its maximum. Then, objects are consumed accordingly. It is in this stage that the flag checking for each block is performed.

Algorithm 5.2 Initialization

Input: $\Pi = (G, \Sigma, \Phi, T, \{(f_j, E_j) \mid 1 \leq j \leq m\}, \mathcal{R}_E)$

$C_0 \leftarrow \{f_1, E_1, \ldots, f_m, E_m\}$ ▷ Initial configuration
for $i \leftarrow 1$ **to** m **do** ▷ For each cell
  $B_i \leftarrow$ ordered set of blocks formed by rules of $\mathcal{R}$ associated with cell i
  $\alpha_i \leftarrow |B_i|$ ▷ Number of rule blocks
  for $j \leftarrow 1$ **to** α_i **do** ▷ For each block associated with the cell
    $B_{i,j} \leftarrow$ ordered set of rules from j^{th} block in B_i.
    $h_{i,j} \leftarrow |B_{i,j}|$ ▷ Number of rules within the block
    $NBA_{i,j} \leftarrow 0$ ▷ Initially, all blocks applications are 0
    for $k \leftarrow 1$ **to** $h_{i,j}$ **do** ▷ Initially, all rule applications are 0
      $NRA_{i,j,k} \leftarrow 0$
    end for
  end for
end for

Moreover, blocks of type 2 (context–changing rules) consume and generate the new flag.

Algorithm 5.3 Phase 1: Object distribution among blocks

Input: $C'_t = \{x_{1,t}, u_{1,t}, \ldots, x_{m,t}, u_{m,t}\}$

for $i \leftarrow 1$ **to** m **do** ▷ For each cell
  for $j \leftarrow 1$ **to** α_i **do** ▷ For each block associated with the cell
    if $Flag(B_{i,j}) = x_{i,t}$ **then**
      if $Type(B_{i,j}) = 1 \wedge Mult(B_{i,j}) \subseteq u_{i,t}$ **then**
        $NBA_{i,j} \leftarrow \min(\lfloor \frac{u_{i,t}(z)}{Mult(B_{i,j})(z)} \rfloor : z \in \Sigma)$ ▷ Maximal application
        $u_{i,t} \leftarrow u_{i,t} - NBA_{i,j} \cdot Mult(B_{i,j})$ ▷ Update the configuration
      end if
      if $Type(B_{i,j}) = 2 \wedge Mult(B_{i,j}) \subseteq u_{i,t}$ **then**
        $NBA_{i,j} \leftarrow 1$ ▷ Just one application
        $x_{i,t} \leftarrow g$, being $RHS(r_{i,j,1}) = (i, g, v)$ with $B_{i,j} = \{r_{i,j,1}\}$ ▷ Update cell flag
        $u_{i,t} \leftarrow u_{i,t} - NBA_{i,j} \cdot Mult(B_{i,j})$ ▷ Update the configuration
      end if
    end if
  end for
end for

Next, objects are distributed among rules according to a binomial distribution with rule probabilities and maximum number of block applications as parameters. This algorithm is composed of two stages *non–maximal* and *maximal* repartition. In the non–maximal repartition stage, a rule in the block is randomly selected according to a uniform distribution, so each rule has the same probability to be chosen. Then, its number of applications is calculated according to an *ad–hoc*

procedure based on a binomially distributed variable $Binomial(n, p)$, where n is the remaining number of block applications to be assigned among its rules and p is the corresponding rule probability. This process is repeated a number K of iterations for each block $B_{i,j}, 1 \leq i \leq m, 1 \leq j \leq \alpha_i$. Algorithm 5.4 describes this procedure. If, after this process, there are still applications to assign among rules, a rule per applicable block is chosen at random and as many applications as possible are assigned to it in the maximal repartition stage. An alternative approach would be to implement a multinomial distribution of applications for the rules inside each block, such as the way that it is implemented on the DCBA algorithm [15]. A method to implement a multinomial distribution would be the conditional distribution method, which emulates a multinomial distribution based on a sequence of binomial distributions [8]. This would require to normalize rule probabilities for each rule application distribution iteration. This approach has also been tested on the simulation algorithm, but was discarded because it tends to distribute too few applications in the non–maximal repartition stage, thus leaving too many applications for the rule selected in the maximal repartition one.

Algorithm 5.4 Phase 2: Rule application distribution

Input: $C'_t = \{x_{1,t}, u_{1,t}, \ldots, x_{m,t}, u_{m,t}\}$
  for $k \leftarrow 1$ **to** K **do** ▷ Non-maximal repartition stage
    for $i \leftarrow 1$ **to** m **do**
      for $j \leftarrow 1$ **to** α_i **do**
        $l \leftarrow Uniform\{1, \ldots, h_{i,j}\}$ ▷ Select a random rule $r_{i,j,l}$ in Block $B_{i,j}$
        $lnrap \leftarrow Binomial(NBA_{i,j}, p_{\mathcal{R}}(r_{i,j,l}))$
        $NRA_{i,j,l} \leftarrow NRA_{i,j,l} + lnrap$ ▷ Update rule applications
        $NBA_{i,j} \leftarrow NBA_{i,j} - lnrap$
      end for
    end for
  end for
  for $i \leftarrow 1$ **to** m **do** ▷ Maximal repartition stage
    for $j \leftarrow 1$ **to** α_i **do**
      $l \leftarrow Uniform\{1, \ldots, h_{i,j}\}$
      $NRA_{i,j,l} \leftarrow NRA_{i,j,l} + NBA_{i,j}$
      $NBA_{i,j} \leftarrow 0$
    end for
  end for

Lastly, rules produce objects as indicated by their right–hand side. Each rule produces objects according to its previously assigned number of applications. Algorithm 5.5 describes this procedure.

The algorithm proposed in this paper works only for models without object competition. This is because the models studied so far did not have object competition, so this feature was not required. However, it might be interesting to develop new algorithms supporting it. They would be identical to their counterpart

Algorithm 5.5 Phase 3: Object production

for $i \leftarrow 1$ **to** m **do** ▷ For each cell
  for $j \leftarrow 1$ **to** α_i **do** ▷ For each block associated with the cell
    for $k \leftarrow 1$ **to** $h_{i,j}$ **do** ▷ For each rule belonging to the block
      $u_{i,t} \leftarrow u_{i,t} + NRA_{i,j,k} \cdot v$, where $RHS(r_{i,j,k}) = (i', f', v)$
      $NRA_{i,j,k} \leftarrow 0$
    end for
  end for
end for

without object competition, solely differing in the protocol by which objects are distributed among blocks. As an example, it would be possible to adapt the way in which objects are distributed in the DCBA algorithm [15].

5.3 Software environment

Next, the developed simulators, a P–Lingua extension, and a GUI for PGP systems are going to be summarized.

Simulators

A simulator for PGP systems without object competition has been incorporated on P–Lingua [10]. In addition, a C++ simulator for PGP systems (namely PGPC++) has also been implemented. The libraries used for random number generation are COLT [23] in the P–Lingua simulator, and standard std::rand [24] for PGPC++. In the latter, the facilities provided by std::rand are directly used. These libraries provide a wide range of functionality to generate and handle random numbers, and are publicly available under open source licenses.

P–Lingua extension

In order to define PGP systems, P–Lingua has been extended to support PGP rules. Specifically, given $f, g \in \Phi,\ u, v \in M(\Sigma),\ 1 \leq i, j \leq m,\ p = p_{\mathcal{R}}(r)$, rules are represented as follows:

$$\{f\}\,(u)_i \xrightarrow{p} (v)_j,\ \equiv \texttt{@guard } f \texttt{ ?[}u\texttt{]'}i \texttt{ --> [}v\texttt{]'}j \texttt{ :: } p\texttt{;}$$
$$\{f\}\,(u, f)_i \to (v, g)_i \equiv \texttt{@guard } f \texttt{ ?[}u\texttt{,}f\texttt{]'}i \texttt{ --> [}v\texttt{,}g\texttt{]'}i \texttt{ :: 1.0;}$$

In both cases, if $p = 1.0$, then `:: p` can be omitted. If $i = j$, then $\{f\}\,(u)_i \xrightarrow{p} (v)_j$ can be written as `@guard f ?[u --> v]'i :: p;`. Likewise, $\{f\}\,(u, f)_i \to (v, g)_i$ can always be written as `@guard f ?[u,f --> v,g]'i;`. Moreover, some additional constructs have been included to ease parametrization of P systems. The idea is to enable completely parametric designs, so as experiments can be tuned by simply adjusting parameters, leaving modifications of P–Lingua files for cases in which changes in semantics are in order.

`&{`*multiset*`}:{`*iterators*`}` In this sentence, *multiset* is an ordinary multiset, whose indexes depend on the iterators defined in *iterators*. *iterators* is a standard list of iterators in P–Lingua separated by commas. It is worth noting that this sentence has some limitations. For instance, variables defined in these iterators cannot be used again in the same P–Lingua specification. In addition, those variables used in *multiset* which are defined in *iterators* can only be used as such, that is, they cannot be used as subindexes or arithmetical expressions. The reasons for these constraints correspond to technical implementation details which will not be discussed here.

`@mu(`*label*`)*=`*cell_structure*`;` In this sentence, *label* is a cell label defined at some point in the P–Lingua specification. *cell_structure* is a standard P–Lingua, tissue–like membrane structure, such as the ones which can be defined after the *@mu* sentence. This sentence adds the skin of *membrane_structure* as a child cell of *label*. As cells in tissue–like structures have no parent, $label = 0$ for all tissue–like models. In cell–like models, the behaviour is the same, with the exception that *cell_structure* is a cell–like structure, *label* can be any label in the system and the symbol `*=` is replaced by `+=`.

`@property(`*label*`)=`*set*`;` This sentence allows designers to define specific properties for objects. *set* is a set of symbols, which can be extended by external, standard iterators or internal ones as defined at the first point of this list. In the case of PGP systems, @property(flag)=*set* defines flags $f \in \Phi$.

In addition, two new formats have been integrated into P-Lingua. These formats (XML–based and binary) encode P systems representing labels and objects as numbers instead of strings, so they are easily parsed and simulated by third–part simulators such as PGPC++.

A graphical environment for PGP systems

MeCoGUI is a new GUI developed for the simulation of PGP systems. MeCoSim [20] could have been used instead. However, in the environment in which the simulators were developed there exist some pros and cons on this approach versus and ad–hoc simulator.

MeCoSim is an integrated development environment (IDE). That is to say, it provides all functionality required for the simulation and computational analysis of P systems. To define the desired input and output displays, it is necessary to configure a spreadsheet by using an *ad–hoc* programming language. However, it would entail teaching this language to prospective users, which are proficient in R programming language instead. In this sense, a more natural approach for them is to develop a GUI in which users can define input parameters and results analysis on R.

To do so, the developed GUI takes as input a P system file on P–Lingua format and a CSV file encoding its parameters, and outputs a CSV file which contains simulation results. This way, users can define inputs and analyse outputs on their programming language of choice. CSV is a widespread, simple and free

format with plenty of libraries for different languages. This flexibility comes at the cost concerning that the developed GUI is not an IDE, as input parameters and simulation analysis cannot be directly input and viewed on the GUI. Rather, it is necessary to develop applications to generate and process these CSV files which depend on the domain of use. In some simulators (such as PGPC++), the output CSV files represent labels and objects as integers, but this application includes a button to translate output files from PGPC++ into string–representative file formats. Figure 3 displays the main screen of this application.

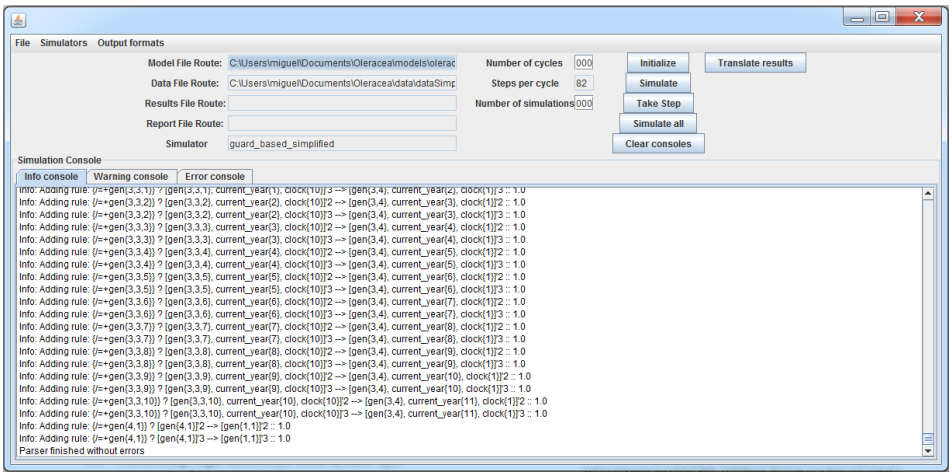


Fig. 3: Main screen of MeCoGUI

MeCoGUI can also translate P systems into machine–readable formats, such as those read by PGPC++. Finally, it is important to remark that these applications play the role of domain–specific spreadsheets on MeCoSim, so MeCoGUI can simulate any type of P system supported by P–Lingua. This is because only external applications for input data and simulation processing depend on the domain, not MeCoGUI itself, which is general for any type of P system. Figure 4 graphically describes the workflow for P–Lingua and for PGPC++.

6 Applications of PGP systems

A model of the ecosystem of the white cabbage butterfly (*Pieris oleracea*) [7], based on PGP systems, is a currently ongoing project. Such a species is suffering the invasion of the garlic mustard (*Alliaria petiolata*), which is replacing native host broadleaf toothwort (*Cardamine diphylla*) and ravaging the butterfly's natural habitat. Specifically, *A. petiolata* contains a deterrent agent for larvae of *P. oleracea*. Moreover, such a plant is toxic for these larvae, although it contains

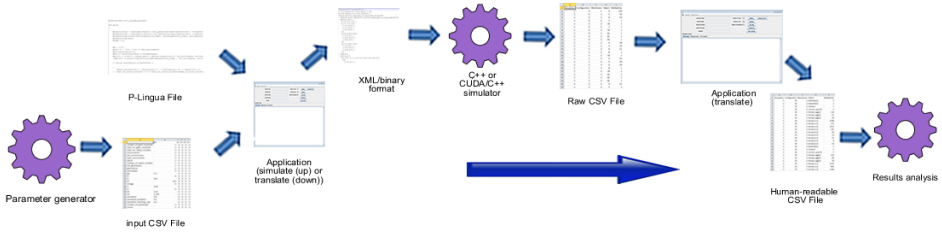


Fig. 4: Workflow for P-Lingua simulator (upper branch) and PGPC++ (lower branch) for MeCoGUI

a chemical compound which lures mature butterflies and frames them into laying eggs. Nevertheless, a minority of individuals tolerates such a deterrent, metabolize the toxin and reach the pupa stage [12, 13].

The distribution of phylogenetic profiles across the species consists of a majority of homozygous individuals unable to thrive on *A. petiolata* patches, a minority of homozygous individuals which do well on *A. petolata* rosettes and, in the midterm, an slightly larger population of heterozygous individuals with both alleles. The allele which enables butterflies to overcome the dietary restrictions imposed by *A. petiolata* is dominant, but individuals carrying this allele undergo a detoxification mechanism which entails an energetic cost and hampers their arrival at adulthood [12].

The model under development aims to identify if there has been any evolutionary adaptation of the butterfly species significant enough so as to ensure its survival in the new scenario. Specifically, the idea is to assess if the detoxification cost associated with individuals tolerating *A. petiolata* pays off in the new scenario or, on the other hand, the phylogenetic distribution will stay the same and other mechanism will come into effect, such as hybridization with other butterfly species such as *Pieris rapae* [7].

The approach taken in this project aims to validate the model *qualitatively*. A *qualitative validation* is defined as follows: a model is qualitatively validated if it can reproduce some properties verified by the ecosystem under different scenarios (according to the experts).

7 Conclusions and Future Work

Multienvironment P systems are a general, formal framework for modelling population dynamics in Biology. The framework has two main approaches: stochastic (micro–level oriented) and probabilistic (macro–level oriented). The framework has been extended in the probabilistic approach, with the inclusion of a new modelling framework called Probabilistic Guarded P (PGP) systems. PGP systems are inspired by Population Dynamics P systems, and aim to simplify the

design and simulation of models of ecological phenomena. The model has been formalized in this paper, and a simulation algorithm is introduced. This algorithm is restricted for models which do not feature object competition. Moreover, an extension of the P–Lingua language is provided to enable PGP systems in P–Lingua, as well as a Graphical User Interface (GUI) to simulate PGP systems (MeCoGUI).

The framework of PGP systems is being utilised for modelling the ecosystem of *Pieris napi oleracea*, a butterfly native to Northeaster U.S.A. The aim is to validate the model qualitatively; that is, checking that if the ecosystem verifies some properties under different scenarios (experts), our model reproduces those properties as well.

Although PGP systems provide a simplified alternative to PDP systems, some constraints to the supported models are imposed: only models without object competition are allowed. Therefore, future research lines will be focused on overcoming this constraint, providing new simulation algorithms permitting object competition. Moreover, new case studies will be considered, what can help to extend the framework. Finally, PGP simulation will be accelerated by using parallel architectures, such as GPU computing with CUDA.

Acknowledgements

The authors acknowledge the support of the project TIN2012-37434 of the "Ministerio de Economía y Competitividad" of Spain, co-financed by FEDER funds. Manuel García–Quismondo also acknowledges the support from the National FPU Grant Programme from the Spanish Ministry of Education. Miguel A. Martínez-del-Amor also acknowledges the support of the 3rd Postdoctoral phase of the PIF program associated with "Proyecto de Excelencia con Investigador de Reconocida Valía" of the "Junta de Andalucía" under grant P08-TIC04200.

References

1. J. Blakes, J. Twycross, F.J. Romero-Campero, N. Krasnogor. The Infobiotics Workbench: an integrated in silico modelling platform for Systems and Synthetic Biology, *Bioinformatics*, **27**, 23 (2011), 3323-3324.
2. M.A. Colomer-Cugat, M. García-Quismondo, L.F. Macías-Ramos, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. PérezJiménez, A. Riscos-Núñez, L. Valencia-Cabrera. Membrane system-based models for specifying Dynamical Population systems. In P. Frisco, M. Gheorghe, M.J. Prez-Jimnez (eds.), *Applications of Membrane Computing in Systems and Synthetic Biology. Emergence, Complexity and Computation series*, Volume **7**. Chapter 4, pp. 97–132, 2014, Springer Int. Publishing.
3. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A computational modeling for real ecosystems based on P systems, *Natural Computing*, **10**, 1 (2011), 39–53.

4. S. Cheruku, A. Păun, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems, *Progress in Natural Science*, **17**, 4 (2007), 424–431.
5. M.A. Colomer, A. Margalida, D. Sanuy, M.J. Pérez-Jiménez. A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study, *Ecological modelling*, **222**, 1 (2011), 33–47.
6. M.A. Colomer, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. A uniform framework for modeling based on P Systems. *Proceedings IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010)*, **Volume 1**, pp. 616–621.
7. F.S. Chew. Coexistence and local extinction in two pierid butterflies, *The American Naturalist*, **118**, 5 (1981), 655–672.
8. C.S. Davis. The computer generation of multinomial random variates. *Computational Statistics and Data Analysis*, **16**, 2 (1993), 205–217.
9. P. Frisco, M. Gheorghe, M. J. Pérez-Jiménez (eds.) *Applications of Membrane Computing in Systems and Synthetic Biology*, Springer, 2014.
10. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Agustín Riscos-Núñez. An overview of P-Lingua 2.0, *LNCS*, **5957** (2010), 264–288.
11. M. Gheorghe, F. Ipate, C. Dragomir, L. Mierla, L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez. Kernel P systems - Version I, *Proceedings of the Eleventh Brainstorming Week on Membrane Computing (BWMC2013)*, 2013, pp. 97–124.
12. M.S. Keeler, F.S. Chew. Escaping an evolutionary trap: preference and performance of a native insect on an exotic invasive host, *Oecologia*, **156**, 3 (2008), 559–568.
13. M.S. Keeler, F.S. Chew, B.C. Goodale, J.M. Reed. Modelling the impacts of two exotic invasive species on a native butterfly: top-down vs. bottom-up effects, *Journal of Animal Ecology*, **75**, 3 (2006), 777–788.
14. M.A. Martínez-del-Amor, I. Pérez-Hurtado, A. Gastalver-Rubio, A.C. Elster, M.J. Pérez-Jiménez. Population Dynamics P systems on CUDA. *LNBI*, **7605** (2012), 247–266.
15. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Romero-Jiménez, C. Graciani, A. Riscos-Núñez, M.A. Colomer, M.J. Pérez-Jiménez. DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution, *LNCS*, **7762** (2012), 27–56.
16. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, F. Sancho-Caparrini. A simulation algorithm for multienvironment probabilistic P systems: A formal verification, *International Journal of Foundations of Computer Science*, **22**, 1 (2011), 107–118.
17. A. Păun, B. Popa. P systems with proteins on membranes, *Fundamenta Informaticae*, **72**, 4 (2006), 467–483.
18. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and TUCS Report No 208.
19. G. Păun, G. Rozenberg, A. Salomaa (eds.). *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
20. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez. MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems, *Proceedings IEEE Fifth International*

Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010), **volume I** (2010), pp. 637–643.

21. M.J. Pérez-Jiménez, F.J. Romero. P systems. A new computational modelling tool for Systems Biology. *Transactions on Computational Systems Biology VI. Lecture Notes in Bioinformatics*, **4220** (2006), 176–197.
22. L. Pan, M.J. Pérez-Jiménez. Computational complexity of tissue-like P systems, *Journal of Complexity*, **26**, 3 (2010), 296–315.
23. *COLT library*. `http://acs.lbl.gov/software/colt/index.html`
24. *RAND function in C++/C Standard General Utilities Library (cstdlib)*. `http://www.cplusplus.com/reference/cstdlib/rand`

Solving the ST-Connectivity Problem with Pure Membrane Computing Techniques

Zsolt Gazdag[1], Miguel A. Gutiérrez–Naranjo[2]

[1]Department of Algorithmics and their Applications
Faculty of Informatics
Eötvös Loránd University, Hungary
gazdagzs@inf.elte.hu

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
magutier@us.es

Summary. In Membrane Computing, the solution of a decision problem X belonging to the complexity class **P** via a polynomially uniform family of recognizer P systems is trivial, since the polynomial encoding of the input can involve the solution of the problem. The design of such solution has one membrane, two objects, two rules and one computation step. *Stricto sensu*, it is a solution in the framework of Membrane Computing, but it does not use Membrane Computing strategies. In this paper, we present three designs of uniform families of P systems that solve the decision problem STCON by using Membrane Computing strategies (*pure* Membrane Computing techniques): P systems with membrane creation, P systems with active membranes with dissolution and without polarizations and P systems with active membranes without dissolution and with polarizations. Since STCON is **NL**-complete, such designs are *constructive* proofs of the belonging of **NL** to $\mathbf{PMC}_{\mathcal{MC}}$, $\mathbf{PMC}_{\mathcal{AM}^{0}_{+d}}$ and $\mathbf{PMC}_{\mathcal{AM}^{+}_{-d}}$.

1 Introduction

Membrane Computing [13] is a well-established model of computation inspired by the structure and functioning of cells as living organisms able to process and generate information. It starts from the assumption that the processes taking place in the compartmental structures as living cells can be interpreted as computations. The devices of this model are called *P systems*.

Among the different research lines in Membrane Computing, one of the most vivid is the search of frontiers between complexity classes of decision problems, i.e., to identify collections of problems that can be solved (or languages that can be decided) by families of P systems with *similar* computational resources. In order to settle the correspondence between complexity classes and P system families,

recognizer P systems were introduced in [9, 10]. Since then, recognizer P systems are the natural framework to study and solve decision problems within Membrane Computing.

In the last years, many papers have been published about the problem of deciding if a uniform family of recognizer P systems of type $\mathcal{F}$ built in polynomial time is able to solve the decision problem X . This is usually written as the problem of deciding if X belongs to $\mathbf{PMC}_{\mathcal{F}}$ or not. It has been studied for many P system models $\mathcal{F}$ and for many decision problems X (see, e.g., [2, 3, 4, 5] and references therein).

The solution of a decision problem X belonging to the complexity class $\mathbf{P}$ via a polynomially uniform family of recognizer P systems is trivial[1], since the polynomial encoding of the input can involve the solution of the problem. On the one hand, by definition, $X \in \mathbf{P}$ if there exists a deterministic algorithm A working in polynomial time that *solves* X. On the other hand, the belonging of X to $\mathbf{PMC}_{\mathcal{F}}$ requires a polynomial time mapping *cod* that encodes the instances u of the problem X as multisets wich will be provided as inputs. Formally, given a decision problem X and an algorithm A as described above, two different functions *s (size)* and *cod (encoding)* can be defined for each instance u of the decision problem:

- $s(u) = 1$, for all u
- $cod(u) = \begin{cases} yes & \text{if } A(u) = yes \\ no & \text{if } A(u) = no. \end{cases}$

The family of P systems which solves X is $\boldsymbol{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ with

$$\Pi(n) = \langle \Gamma, \Sigma, H, \mu, w, \mathcal{R}, i \rangle$$

- *Alphabet:* $\Gamma = \{yes, no\}$
- *Input alphabet:* $\Sigma = \Gamma$
- *Set of labels:* $H = \{skin\}$
- *Membrane structure:* $[\,]_{skin}$
- *Initial multisets:* $w = \emptyset$
- *Input label:* $i = skin$
- *Set of rules:* $[\,yes\,]_{skin} \to yes\,[\,]_{skin}$ and $[\,no\,]_{skin} \to no\,[\,]_{skin}$. Both are send-out rules.

Trivially, for all instance u of the problem, $\Pi(s(u)) + cod(u)$ provides the right solution in one computation step. *Stricto sensu*, it is a solution in the framework of Membrane Computing, but it does not use Membrane Computing strategies. All the work is done in the algorithm A and one can wonder if the computation itself can be performed by using *pure* Membrane Computing techniques.

We focus now on the well-known ST-Connectivity problem (known as STCON). It can be settled as follows: *Given a directed graph* $\langle V, E \rangle$ *and two*

[1] See [8, 11].

vertex s and t in V, the STCON *problem consists on deciding if t is reachable from s, i.e., if there exists a sequence of adjacent vertices (i.e., a path) which starts with s and ends with t.* It is known that it is an **NL**-complete problem, i.e., it can be solved by a nondeterministic Turing machine using a logarithmic amount of memory space and every problem in the class **NL** is reducible to STCON under a log-space reduction.

In this paper, we study the STCON in the framework of P systems. As shown above, since STCON $\in$ **NL** $\subseteq$ **P**, there exist a trivial family of P systems in $\mathbf{PMC}_{\mathcal{F}}$ which solves it, regardless the model $\mathcal{F}$. It suffices that $\mathcal{F}$ deals with *send-out* rules. In this paper, we present three designs of uniform families of P systems that solve the decision problem STCON by *pure* Membrane Computing techniques, i.e., techniques where the features of the model $\mathcal{F}$ are exploited in the computation: P systems with membrane creation, P systems with active membranes with dissolution and without polarizations and P systems with active membranes without dissolution and with polarizations. We provide such designs and show the differences with previous studies found in the literature.

Since STCON is **NL**-complete, such designs are *constructive* proofs of the belonging of **NL** to $\mathbf{PMC}_{\mathcal{MC}}$, $\mathbf{PMC}_{\mathcal{AM}^0_{+d}}$ and $\mathbf{PMC}_{\mathcal{AM}^+_{-d}}$.

The paper is structured as follows: First of all, we recall some basic definitions used along the paper. In Section 3, previous works on **NL** are revisited. Next, our designs of solutions are provided and the paper finishes with some conclusions and presenting research lines for a future work.

2 Preliminaries

Next, some basic concepts used along the paper are recalled. We assume that the reader is familiar with Membrane Computing techniques (for a detailed description, see [13]).

A decision problem, X, is a pair (I_X, θ_X) such that 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. A *P system with input* is a tuple (Π, Σ, i_Π), where Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$ associated with them; Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma - \Sigma$; and i_Π is the label of a distinguished (input) membrane. Let (Π, Σ, i_Π) be a P system with input, Γ be the working alphabet of Π, μ its membrane structure, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$. We denote by I_Π the set of all inputs of the P system Π (i.e. I_Π is a collection of multisets over Σ). In the case of P systems with input and *with external output*, the above concepts are introduced in a similar way.

Definition 1. *A* recognizer P system *is a P system with input, (Π, Σ, i_Π), and with external output such that:*

1. *The working alphabet contains two distinguished elements yes, no.*
2. *All its computations halt.*
3. *If $\mathcal{C}$ is a computation of Π, then either some object yes or some object no (but not both) must have been released into the environment, and only in the last step of the computation. We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object yes (respectively, no) appears in the external environment associated to the corresponding halting configuration of $\mathcal{C}$.*

3 Previous Works

The relation between the complexity class **NL** and Membrane Computing models has already been explored in the literature. In [6], Murphy and Woods claim that $\mathbf{NL} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d,-u}}$, i.e., every problem in the complexity class **NL** can be solved by a non-uniform family of recognizer P systems with active membranes without polarization and without dissolution.

The proof shows the design of a family of P systems with active membranes without polarization and without dissolution which solves STCON and considers the **NL**-completeness of STCON. Nonetheless, the authors use a non standard definition of recognizer P systems. According to the usual definition of *recognizer P system* (see, e.g., [4]), *either one object yes or one object no (but no both) must have been released into the environment, and only in the last step of the computation.* In the proposed family by Murphy and Woods, it is easy to find a P system which sends *yes* to the environment in an intermediate step of the computation and sends *no* to the environment in the last step of the computation, so their proof of $\mathbf{NL} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d,-u}}$ cannot be considered valid with respect to the standard definition of recognizer P systems.

Counterexample: Let us consider the instance (s, t, G) of STCON where G has only two vertices s and t and only one edge (s, t). According to [6], the P system of the cited model that solves this instance has $\Gamma = \{s, t, yes, no, c_0, \ldots, c_4\}$ as alphabet, h as unique label and $[\,]_h$ as membrane structure. The initial configuration is $[s\, c_4]_h$ and the set of rules consists of the following seven rules:

$$[s \to t]_h \qquad [t]_h \to [\,]_h\, yes$$
$$[c_0]_h \to [\,]_h\, no \qquad [c_i \to c_{i-1}]_h \text{ for } i \in \{1, \ldots, 4\}.$$

It easy to check that this P system sends *yes* to the environment in the second step of computation and sends *no* in the fifth (and last) step, so, according to the standard definition, it is not a recognizer P system. In [7] Murphy and Woods revisited the solution of STCON by non-uniform families of recognizer P systems and considered three different ways of the acceptance in recognizer P systems, one of them was the standard one (Def. 1).

4 Three Designs for the STCON Problem

In this section, we provide three uniform families of P systems that solve the STCON problem in three different P system models. All these models use the same encoding for an instance of the problem. We do not loss generality if we consider the n vertices of the graph as $\{1, \ldots, n\}$. In this case, a concrete instance $I = (s, t, \langle V, E\rangle)$ of the STCON on a graph $\langle V, E\rangle$ with vertices $\{1, \ldots, n\}$, can be encoded as

$$cod(I) = \{x_s, y_t\} \cup \{a_{ij} : (i, j) \in E\},$$

i.e., x_s stands for the starting vertex, y_t for the ending vertex and a_{ij} for each edge (i, j) in the graph. By using this coding, *all* the instances of the STCON problem with n variables, can be encoded with the alphabet

$$\begin{aligned}\Sigma = &\{x_i : i \in \{1, \ldots, n\}\} \cup \\ &\{y_j : j \in \{1, \ldots, n\}\} \cup \\ &\{a_{ij} : i, j \in \{1, \ldots, n\}\}\end{aligned}$$

whose cardinality is $2n + n^2$.

Next we present three solutions of the STCON problem by P systems. The first two solutions are based on P systems with active membranes, the last one uses P systems with membrane creation. The first solution does not use membrane dissolution but uses the polarizations of the membranes. The second solution does not use polarizations but uses membrane dissolution instead. Moreover, none of these solutions use membrane division rules.

All the three solutions, roughly speaking, work in the following way. For a given directed graph $G = (V, E)$ and vertices s and t, the system creates/activates certain membranes in the initial configuration corresponding to the edges in E. Then, these membranes will be used to create those objects that represent the vertices reachable from s. Meanwhile, it is tested whether or not the vertex t is created or not. If yes, the system initiates a process which will send *yes* out to the environment. If the vertex t is not produced by the system, i.e., t is not reachable from s in G, then a counter will create the symbol *no* which is then sent out to the environment.

4.1 P Systems with Active Membranes, with Polarization and without Dissolution

As a first approach, we will provide the design of a uniform family $\mathbf{\Pi} = \{\Pi_n\}_{n\in\mathbb{N}}$ of P systems in $\mathbf{PMC}_{\mathcal{AM}_{-d}}$ which solves STCON. Each P system Π_n of the family decides on *all the possible* instances of the STCON problem on a graph with n nodes. Such P systems use two polarizations, but they do not use division or dissolution rules, so *not all the types of rules* of P systems with active membranes are necessary to solve STCON. Each Π_n will receive as input an instance of the STCON as described above and will release *yes* or *no* into the environment in the

last step of the computation as the answer of the decision problem. The family presented here is

$$\Pi_n = \langle \Gamma_n, \Sigma_n, H_n, EC_n, \mu_n, w_n^a, w_n^1, \ldots, w_n^n, w_n^{11}, \ldots, w_n^{nn}, w_n^{skin}, \mathcal{R}_n, i_n \rangle.$$

For the sake of simplicity, thereafter we will omit the subindex n.

- **Alphabet:**

$$\begin{aligned}\Gamma = &\{x_i, y_i, t_i : i \in \{1, \ldots, n\}\} \cup \\ &\{a_{ij}, z_{ij} : i, j \in \{1, \ldots, n\}\} \cup \\ &\{c_i : i \in \{0, \ldots, 3n+1\}\} \cup \\ &\{k, yes, no\}.\end{aligned}$$

- **Input alphabet:** Σ, as described at the beginning of the section. Let us remark that $\Sigma \subset \Gamma$.
- **Set of labels:** $H = \{\langle i, j \rangle : i, j \in \{1, \ldots, n\}\} \cup \{1, \ldots, n\} \cup \{a, skin\}$.
- **Electrical charges:** $EC = \{0, +\}$.
- **Membrane structure:** $[\,[\,]_1^0 \ldots [\,]_n^0\,[\,]_{\langle 1,1 \rangle}^0 \ldots [\,]_{\langle n,n \rangle}^0\,[\,]_a^0\,]_{skin}^0$.
- **Initial multisets:** $w^a = c_0$, $w^{skin} = w^{ij} = w^k = \lambda$ for $i, j, k \in \{1, \ldots, n\}$.
- **Input label:** $i = skin$.

The set of rules $\mathcal{R}$:

R1. $a_{ij}\,[\,]_{\langle i,j \rangle}^0 \to [a_{ij}]_{\langle i,j \rangle}^+$ for $i, j \in \{1, \ldots, n\}$.
Each input object a_{ij} activates the corresponding membrane by changing its polarization. Notice that such a symbol a_{ij} represents an edge in the input graph.
R2. $y_j\,[\,]_j^0 \to [y_j]_j^+$ for $j \in \{1, \ldots, n\}$.
The object y_j activates the membrane j by changing its polarization. As the input multiset always has exactly one object of the form y_j, Π_n will have a unique membrane with label in $\{1, \ldots, n\}$ and polarization $+$.
R3. $[x_i \to z_{i1} \ldots z_{in} t_i]_{skin}^0$ for $i \in \{1, \ldots, n\}$.
The goal of these rules is to create $n+1$ copies of an object x_i. A copy z_{ij} will be able to produce an object x_j if the edge (i, j) belongs to E. The object t_i will be used to witness that vertex i is reachable.
R4.
$$\left.\begin{array}{l} z_{ij}\,[\,]_{\langle i,j \rangle}^+ \to [x_j]_{\langle i,j \rangle}^0 \\ t_j\,[\,]_j^+ \to [k]_j^0 \end{array}\right\} \text{for } i, j \in \{1, \ldots, n\}.$$
If the membrane with label $\langle i, j \rangle$ has polarization $+$, then the symbol z_{ij} produces a symbol x_j inside this membrane. Meanwhile, the polarization of this membrane changes from $+$ to 0, i.e., the membrane is deactivated. Moreover, if the symbol t_j appears in the skin and the membrane with label j has positive polarization, then an object k is produced inside this membrane. Such object k will start the process to send *yes* out to the environment.
R5. $[k]_j^0 \to k\,[\,]_j^0 \qquad k\,[\,]_a^0 \to [k]_a^+$.
The object k is a witness of the success of the STCON problem. If it is produced, it goes into the membrane with label a and changes its polarization to $+$.
R6. $[x_j]_{\langle i,j \rangle}^0 \to x_j\,[\,]_{\langle i,j \rangle}^0$ for $i, j \in \{1, \ldots, n\}$.

The produced object x_j is sent to the membrane *skin* in order to go on the computation by rules form **R3**.

R7. $\left.\begin{array}{l}[c_i \to c_{i+1}]_a^0\ [c_{3n+1}]_a^0 \to no\,[\,]_a^0 \\ [c_i \to c_{i+1}]_a^+\ [c_{3n+1}]_a^+ \to yes\,[\,]_a^0\end{array}\right\}$ for $i \in \{0,\ldots,3n\}$.

Object c_i evolves to c_{i+1} regardless of the polarization of the membrane a. If during the evolution the object k has gone inside such membrane, then the polarization changes to $+$ and the object c_{3n+1} will produce *yes* in the membrane *skin*. Otherwise, if the object k is not produced, the polarization is not changed and the object c_{3n+1} will produce *no*.

R8. $[no]_{skin} \to no\,[\,]_{skin} \qquad [yes]_{skin} \to yes\,[\,]_{skin}$.

Finally, *yes* or *no* is sent out the P system in the last step of computation.

To see in more details how the computation of the presented P system goes, let us consider an instance $I = (s,t,G)$ of STCON where G is a graph $\langle\{1,\ldots,n\},E\rangle$. The computation of Π_n on $cod(I)$ can be described as follows. During the first step, using rules in **R1**, every a_{ij} enters to the membrane with label $\langle i,j\rangle$ and changes its polarization to $+$. Thus, after the first step the edges in E are encoded by the positive polarizations of the membranes with labels of the form $\langle i,j\rangle$. During the same step, using the corresponding rule in **R2**, y_t enters to the membrane with label t and changes its polarization to $+$. This membrane will be used to recognize if an object representing that t is reachable from s is introduced by the system.

Now let $l \in \{1,4,\ldots,3(n-1)+1\}$ and consider an object x_i in the skin membrane. During the lth step, using rules in **R3**, x_i creates $n+1$ copies of itself. The system will try to use a copy z_{ij} ($j \in \{1,\ldots,n\}$) in the next step to create a new object x_j. The copy t_i will be used to decide if $i = t$.

During the $(l+1)$th step, using rules in **R4**, the systems sends z_{ij} into the membrane with label $\langle i,j\rangle$ if that membrane has a positive polarization. Meanwhile, z_{ij} evolves to x_j and the polarization of the membrane changes to neutral. During the same step, if $i = t$ and the membrane with label t has positive polarization, then the system sends t_i to this membrane. Meanwhile, t_i evolves to k and the polarization of membrane t changes to neutral.

During the $(l+2)$th step, using rules in **R6**, the object x_j is sent out from the membrane with label $\langle i,j\rangle$. Moreover, if the membrane with label t contains k, then this k is sent out from membrane t.

One can see that during the above three steps the system introduces an object x_j if and only if (i,j) is an edge in E. Using this observation we can derive that during the computation of the system, an object x_j appears in the skin if and only if there is a path in G from s to j. Thus, t is reachable from s in G if and only if there is a configuration of Π_n where the skin contains x_t. However, in this case an object k is introduced in the membrane with label t. It can also be seen that Π_n sends out to the environment *yes* if and only if k appears in membrane t. Moreover, if k does not appear in membrane t, then the systems sends out to the environment *no*. Thus, Π_n sends out to the environment *yes* or *no* according to that t is reachable from s or not. As Π_n stops in at most $3n+2$ steps, we

can conclude that the family $\mathbf{\Pi}$ decides STCON in linear time in the number of vertices of the input graph.

4.2 P Systems with Active Membranes, with Dissolution and without Polarization

Based on the solution presented in the previous sub-section, we give here a uniform family $\mathbf{\Pi} = \{\Pi_n\}_{n\in\mathbb{N}}$ in $\mathbf{PMC}_{\mathcal{AM}^0}$ which solves STCON. As here we cannot use the polarizations of the membranes, we use membrane dissolution to select those membranes of the initial configuration that correspond to the edges of the input graph. Next we will describe the mentioned family $\mathbf{\Pi}$. Since we do not use polarizations, we do not indicate it at the upper-right corner of the membranes. The family presented here is

$$\Pi_n = \langle \Gamma, \Sigma, H, EC, \mu, W, \mathcal{R}, i\rangle.$$

- **Alphabet:**

$$\begin{aligned}\Gamma = &\{x_i, v_{1i}, v_{2i}, v_{3i}, v_i, y_i, t_i : i \in \{1,\dots,n\}\} \cup \\ &\{a_{ij},\, z_{ij} : i,j \in \{1,\dots,n\}\} \cup \\ &\{c_i : i \in \{0,\dots,3n+4\}\} \cup \\ &\{k, yes, no\}.\end{aligned}$$

- **Input alphabet:** Σ, as described at the beginning of the section.
- **Set of labels:** $H = \{\langle i,j,in\rangle, \langle i,j,out\rangle : i,j \in \{1,\dots,n\}\} \cup \{\langle i,in\rangle, \langle i,out\rangle : i \in \{1,\dots,n\} \cup \{a, skin\}$.
- **Electrical charges:** $EC = \varnothing$.
- **Membrane structure:**
$[\,[\,[\,]_{\langle 1,in\rangle}]_{\langle 1,out\rangle} \cdots [\,[\,]_{\langle n,in\rangle}]_{\langle n,out\rangle}\, [\,[\,]_{\langle 1,1,in\rangle}]_{\langle 1,1,out\rangle} \cdots [\,[\,]_{\langle n,n,in\rangle}]_{\langle n,n,out\rangle}\, [\,]_a\,]_{skin}$.
- **Initial multisets:** $W = \{w^a, w^{\langle 1,in\rangle}, \dots, w^{\langle n,in\rangle}, w^{\langle 1,out\rangle}, \dots, w^{\langle n,out\rangle}, w^{\langle 1,1,in\rangle}, \dots, w^{\langle n,n,in\rangle}, w^{\langle 1,1,out\rangle}, \dots, w^{\langle n,n,out\rangle}, w^{skin}\}$, where $w^a = c_0$, $w^{skin} = w^{\langle i,j,out\rangle} = w^{\langle k,out\rangle} = \lambda$, $w^{\langle i,j,in\rangle} = w^{\langle k,in\rangle} = f_0$, for $i,j,k \in \{1,\dots,n\}$.
- **Input label:** $i = skin$.

The set of rules $\mathcal{R}$:

R0. $[x_i \to v_{1i}]_{skin}$, $[v_{ji} \to v_{j+1,i}]_{skin}$, $[v_{3i} \to v_i]_{skin}$ for $i \in \{1,\dots,n\}$ and $j \in \{1,2\}$.

In this solution we cannot use the objects x_i in the same role as we did in the previous sub-section because of the following reason. The system needs four steps to select those membranes in the initial membrane configuration that correspond to the edges in E. Thus, the system introduces in four steps the objects v_i which will act in this solution as the objects x_i did in the previous one.

R1.
$$\left.\begin{array}{l}[f_m \to f_{m+1}]_{\langle i,j,in\rangle} \\ [f_3]_{\langle i,j,in\rangle} \to f_4 \\ [f_4]_{\langle i,j,out\rangle} \to f_4\end{array}\right\} \text{for } i,j \in \{1,\dots,n\},\ m \in \{0,1,2\}.$$

These rules dissolve the membranes with label $\langle i, j, in\rangle$ and $\langle i, j, out\rangle$ if the input symbol a_{ij} is not present in the system. On the other hand, if a_{ij} is in the system, then it prevents the dissolution of the membrane with label $\langle i, j, out\rangle$ using the following rules.

R2. $\left.\begin{array}{l} a_{ij}\,[\,]_{\langle i,j,m\rangle} \to [a_{ij}]_{\langle i,j,m\rangle} \\ [a_{ij}]_{\langle i,j,in\rangle} \to a_{ij} \end{array}\right\}$ for $i, j \in \{1, \ldots, n\}$, $m \in \{in, out\}$.

By these rules the input symbol a_{ij} goes into the membrane with label $\langle i, j, in\rangle$ and dissolves that. This way the second rule in **R1** cannot be applied, thus the membrane with label $\langle i, j, out\rangle$ cannot be dissolved by the third rule.

R3. $\left.\begin{array}{l} [f_m \to f_{m+1}]_{\langle j,in\rangle} \\ [f_3]_{\langle j,in\rangle} \to f_4 \\ [f_4]_{\langle j,out\rangle} \to f_4 \end{array}\right\}$ for $j \in \{1, \ldots, n\}$, $m \in \{0, 1, 2\}$.

These rules dissolve the membranes with label $\langle j, in\rangle$ and $\langle j, out\rangle$ if the input symbol y_j is not present in the system. However, if y_j is in the system, then it prevents the dissolution of the membrane with label $\langle j, out\rangle$ using the following rules.

R4. $\left.\begin{array}{l} y_j\,[\,]_{\langle j,m\rangle} \to [y_j]_{\langle j,m\rangle} \\ [y_j]_{\langle j,in\rangle} \to y_j \end{array}\right\}$ for $j \in \{1, \ldots, n\}$ and $m \in \{in, out\}$.

By these rules the input symbol y_j goes into the membrane with label $\langle j, in\rangle$ and dissolves that. With this it is achieved that the membrane with label $\langle j, out\rangle$ is not dissolved by the rules in **R3**.

R5. $[v_i \to z_{i1} \ldots z_{in} t_i]_{skin}$ for $i \in \{1, \ldots, n\}$.

The role of these rules is the same as that of the rules in **R3** in Section 4.1.

R6. $\left.\begin{array}{l} z_{ij}\,[\,]_{\langle i,j,out\rangle} \to [v_j]_{\langle i,j,out\rangle} \\ t_j\,[\,]_{\langle j,out\rangle} \to [k]_{\langle j,out\rangle} \end{array}\right\}$ for $i, j \in \{1, \ldots, n\}$.

The role of these rules is similar to that of the rules in **R4** in Section 4.1: If the membrane with label $\langle i, j, out\rangle$ has not been dissolved, then the object z_{ij} produces a symbol v_j inside this membrane. Analogously, if the symbol t_j appears in the skin and the membrane with label $\langle j, out\rangle$ is not dissolved, then an object k is produced inside this membrane. Such object k will start the process to send *yes* out to the environment.

R7. $[k]_{\langle j,out\rangle} \to k\,[\,]_{\langle j,out\rangle} \qquad k\,[\,]_a \to [k]_a \qquad [k]_a \to k.$

The object k is a witness of the success of the STCON problem. If it is produced, it goes into the membrane with label a and dissolves it.

R8. $[v_j]_{\langle i,j\rangle} \to v_j$ for $i, j \in \{1, \ldots, n\}$.

The produced object v_j dissolves the membrane with label $\langle i, j\rangle$ as the computation does not need any more this membranes. This way the object v_j gets to the skin and the computation can go on using the rules in **R5**.

R9. $\left.\begin{array}{l} [c_i \to c_{i+1}]_a \; [c_{3n+4}]_a \to no\,[\,]_a \\ \qquad\qquad\qquad\quad [c_{i+1}]_{skin} \to [yes]_{skin} \end{array}\right\}$ for $i \in \{0, \ldots, 3n + 3\}$.

Object c_i evolves to c_{i+1} in membrane with label a. If during the evolution the object k has gone inside this membrane, then it dissolves it and the object c_{i+1} gets to the membrane s where it produces *yes*. Otherwise, if the object k is not produced, c_{3n+4} remains in membrane with label a and produces *no*.

R10. $[no]_{skin} \rightarrow no\,[\,]_{skin} \qquad [yes]_{skin} \rightarrow yes\,[\,]_{skin}$.
Finally, *yes* or *no* is sent out the P system in the last step of computation.

One can observe that during the first four steps of Π_n a membrane with label $\langle i, j, out\rangle$ is not dissolved if and only if a_{ij} is in the input. Thus, Π_n has a membrane with label $\langle i, j, out\rangle$ after the first four steps if and only if Π_n defined in Section 4.1 has a membrane $\langle i, j\rangle$ with positive polarization after the first step. Similar observations apply in the case of membranes with label $\langle j, out\rangle$. Thus, the correctness of Π_n defined in this section follows from the correctness of Π_n defined in Section 4.1. One can also observe that Π_n stops after at most $3n+5$ steps, which means that the family $\mathbf{\Pi}$ defined in this section decides STCON in linear time.

4.3 P Systems with Membrane Creation

Here we provide the design of a uniform family of P systems in the framework of *P systems with Membrane Creation* which solves the problem STCON. Since STCON is **NL**-complete, we have a direct proof of $\mathbf{NL} \subseteq \mathbf{PMC}_{\mathcal{MC}}$. This result is well-know, since $\mathbf{NL} \subset \mathbf{NP}$ and $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{MC}}$ (see [4]). Nonetheless, to the best of our knowledge, this is the first design of a P system family which solves STCON in $\mathbf{PMC}_{\mathcal{MC}}$.

Next we will describe the family $\mathbf{\Pi} = \{\Pi_n\}_{n\in\mathbb{N}}$ of P systems in $\mathbf{PMC}_{\mathcal{MC}}$. Each Π_n will receive as input an instance of the STCON as described at the beginning of the section and will release *yes* or *no* into the environment in the last step of the computation as the answer of the decision problem.

The family presented here is

$$\Pi_n = \langle \Gamma, \Sigma, H, \mu, w^a, w^b, w^c, \mathcal{R}, i\rangle.$$

- **Alphabet:**

$$\begin{aligned}\Gamma = &\{x_i, y_i, t_i \,:\, i \in \{1,\ldots,n\}\} \cup \\ &\{a_{ij},\, z_{ij} \,:\, i,j \in \{1,\ldots,n\}\} \cup \\ &\{no_i \,:\, i \in \{0,\ldots,3n+3\}\} \cup \\ &\{yes_i \,:\, i \in \{1,\ldots,4\}\} \cup \\ &\{yes, no\}.\end{aligned}$$

- **Input alphabet:** Σ, as it is described at beginning of the section.
- **Set of labels:** $H = \{\langle i, j\rangle \,:\, i, j \in \{1,\ldots,n\}\} \cup \{1,\ldots,n\} \cup \{a, b, c\}$.
- **Membrane structure:** $[\,[\,]_a\,[\,]_b\,]_c$.
- **Initial multisets:** $w^a = no_0$, $w^b = w^c = \lambda$.
- **Input label:** $i = b$.

The set of rules $\mathcal{R}$:

R1. $[[a_{ij} \rightarrow [\,\lambda\,]_{\langle i,j\rangle}\,]_b$ for $i, j \in \{1,\ldots,n\}$.
Each input symbol a_{ij} creates a new membrane with label $\langle i, j\rangle$. Recall that such a symbol a_{ij} represents an edge in the directed graph.

R2. $[y_j \to [\,\lambda\,]_j\,]_b$ for $j \in \{1,\ldots,n\}$.
By these rules an input symbol y_j creates a new membrane with label j.
R3. $[x_i \to z_{i1}\ldots z_{in}t_i]_b$ for $i \in \{1,\ldots,n\}$.
The role of these rules is the same as those of the rules in **R3** in Section 4.1.
R4. $\left.\begin{array}{l} z_{ij}\,[\,]_{\langle i,j\rangle} \to [x_j]_{\langle i,j\rangle} \\ t_j\,[\,]_j \to [yes_0]_j \end{array}\right\}$ for $i,j \in \{1,\ldots,n\}$.
The role of these rules is similar to that of the rules in **R4** in Section 4.1 except that here an object t_j introduces an object yes_0 in the membrane with label j. This new object yes_0 will evolve with the rules in **R6** and **R7** until the final object yes is produced in the environment.
R5. $[x_j]_{\langle i,j\rangle} \to x_j$ for $i,j \in \{1,\ldots,n\}$.
The object x_j dissolves the membrane with label $\langle i,j\rangle$. The useful information is that x_j is reachable. We keep this information, but the membrane can be dissolved. This way x_j gets to the membrane b and the computation can go on using the rules in $\mathbf{R_3}$.
R6. $[yes_0]_j \to yes_1$ for $j \in \{1,\ldots,n\}$.
For each possible value of j, if yes_0 is produced, the corresponding membrane is dissolved and yes_1 appears in the membrane with label b.
R7. $[yes_1]_b \to yes_2$, $yes_2\,[\,]_a \to [yes_3]_a$,
$[yes_3]_a \to yes_4$, $[yes_4]_c \to yes\,[\,]_c$.
The evolution of the objects yes_i firstly produces the dissolution of the membrane b. If this membrane is dissolved, the rules from **R3** will be no longer applied. In a similar way, object yes_3 also dissolves membrane a and this stops the evolution of the objects inside such membrane.
R8. $[no_i \to no_{i+1}]_a$ for $i \in \{1,\ldots,3n+2\}$.
The object no_i evolves inside the membrane a. If this evolution is not halted by the dissolution of the membrane a, these objects will produce the object no in the environment.
R9. $[no_{3n+3}]_a \to no \qquad [no]_c \to no\,[\,]_c$.
If the evolution of no_i is not stopped, the object no_{3n+3} dissolves the membrane a and creates a new object no. This object will be sent to the environment in the next step of the computation.

It is not difficult to see using the comments given after the rules that this solution works essentially in the same way as our first solution. The main difference is that while in Section 4.1 an input symbol a_{ij} is used to change the polarization of a membrane $\langle i,j\rangle$, here this symbol is used to create such a membrane. Thus, the correctness of the solution presented here can be seen using the correctness of the solution given in Section 4.1. It is also clear that the P systems presented here work in linear time in the number of vertices in the input graph.

As we have mentioned, in solutions of problems in **P** via uniform families of P systems it is important to use such input encoding and P system constructing devices that are not capable to compute the correct answer. It is easy to see that the decision processes in the solutions of STCON presented in this paper are

entirely done by the P systems themselves. Thus our solutions could be easily modified so that the construction of the used families and the computation of the input encoding can be carried out by reasonable weak computational devices, for example, by logarithmic-space deterministic Turing machines.

5 Conclusions

The design of a uniform family of recognizer P systems working in polynomial time which solves a decision problem with *pure* Membrane Computing techniques is a hard task, regardless the complexity class of the problem. The difficulty comes from the hard restrictions imposed to such family. Firstly, the use of *input* P systems implies that each instance of the problem must be encoded as a multiset and such multiset must be introduced at the starting configuration in *one* input membrane. The multiset encoding the instance cannot be distributed in several membranes in the starting configuration. Secondly, in *uniform* families, each P system must solve *all* the instances of the problem of the same *size* (regardless of whether the answer is positive or not). This means that the set of rules which leads to send *yes* to the environment and the set of rules which leads to send *no* must be present in the design of the P system; and thirdly, the standard definition of recognizer P systems claims that an object *yes* or *no* (but no both) is sent to the environment in the *last* step of computation.

A deep study of these constraints shows that it is not sufficient to implement a design of P system with the control scheme "*if* the restrictions of the decision problem are satisfied, *then* an object *yes* must be sent to the environment". Instead of such scheme, the design must consider the following structure: "*if* the restrictions are satisfied, *then* an object *yes* must be sent to the environment, *else* an object *no* must be sent". This scheme *if-then-else* must be controlled with the ingredients of the P system model. In the three presented designs, this *if-then-else* scheme is implemented via dissolution, polarization, or membrane creation.

These ideas lead us to consider the necessity of revisiting the complexity classes under **P** and adapt the definition of recognizer P systems for these classes. Some papers in this new research line can be found in the literature (see, e.g., [12]), but further research is needed.

Acknowledgements

This work was partially done during Zsolt Gazdag's visit at the Research Institute of Mathematics of the University of Sevilla (IMUS) partially supported by IMUS. Miguel A. Gutiérrez–Naranjo acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

1. Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.): Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers, Lecture Notes in Computer Science, vol. 7762. Springer (2013)
2. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A Linear Time Solution to the Partition Problem in a Cellular Tissue-Like Model. Journal of Computational and Theoretical Nanoscience 7(5), 884–889 (MAY 2010)
3. Gazdag, Z., Kolonits, G.: A new approach for solving SAT by P systems with active membranes. In: Csuhaj-Varjú et al. [1], pp. 195–207
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. Theoretical Computer Science 371(1-2), 54–61 (2007)
5. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 336–352. Springer, Berlin Heidelberg (2007)
6. Murphy, N., Woods, D.: A characterisation of NL using membrane systems without charges and dissolution. In: Calude, C.S., Costa, J.F., Freund, R., Oswald, M., Rozenberg, G. (eds.) Unconventional Computing, Lecture Notes in Computer Science, vol. 5204, pp. 164–176. Springer Berlin Heidelberg (2008)
7. Murphy, N., Woods, D.: On acceptance conditions for membrane systems: characterisations of L and NL. In: Proceedings International Workshop on The Complexity of Simple Programs. Cork, Ireland, 6-7th December 2008. pp. 172–184. Electronic Proceedings in Theoretical Computer Science. Vol 1 (2009)
8. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun et al. [13], pp. 302 – 336
9. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. In: Csuhaj-Varjú, E., Kintala, C., Wotschke, D., Vaszyl, G. (eds.) Proceeding of the 5th Workshop on Descriptional Complexity of Formal Systems. DCFS 2003. pp. 284–294 (2003)
10. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics 11(4), 423–434 (2006)
11. Porreca, A.E.: Computational Complexity Classes for Membrane System. Master's thesis, Univertita´ di Milano-Bicocca, Italy (2008)
12. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú et al. [1], pp. 342–357
13. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)

Conventional Verification for Unconventional Computing: a Genetic XOR Gate Example

Savas Konur[1], Marian Gheorghe[1], Ciprian Dragomir[1], Florentin Ipate[2], Natalio Krasnogor[3]

[1] Department of Computer Science, University of Sheffield, UK
{s.konur,m.gheorghe,c.dragomir}@sheffield.ac.uk
[2] Department of Computer Science, University of Bucharest
florentin.ipate@ifsoft.ro
[3] School of Computing Science, Newcastle University, UK
natalio.krasnogor@newcastle.ac.uk

Summary. As unconventional computation matures and non-standard programming frameworks are demonstrated, the need for formal verification will become more prevalent. This is so because "programming" in unconventional substrates is difficult. In this paper we show how conventional verification tools can be used to verify unconventional programs implementing a logical XOR gate.

1 Introduction

Unconventional computing, with many aspects including implementations in vivo, vitro and silico, models and methods, programming paradigms and tools, is a rapidly growing research area with results, promises and huge hope in building new computational devices and tools for solving better or/and faster increasingly complex problems than current machines, models and tools, which either produce inefficienct results or are just unable to solve them.

One specific class of models and experiments related to unconventional computing, often called natural computing, is inspired by natural processes occurring in biology or produces in vitro (DNA strands) or in vivo (bacteria) experiments simulating different computational devices. A thorough account of the developments in the area can be found in [1], but we also mention some specific demonstrations of unconventional computing using liposomes [2], programmable polymers [3] and photochromic molecules [4].

XOR gate is a classic computer science concept with various unconventional computing incarnations. More recently some implementations have been provided [5, 6] and solutions using synthetic biology computational tools have been proposed [7].

In this paper we aim to reconsider this problem and to provide a set of unconventional computing models based on the P systems computational paradigm [8]. Here, we consider *stochastic P systems* [9] and *kernel P systems* [10] as representative classes of such models. These models are associated with verification methods using model checking approaches.

The key contributions of the paper are: the *introduction of a set of unconventional models based on P systems*, which naturally describe the genetic XOR gate problem, and the *use of some model checkers for verifying properties of the models.* This approach is complementary to the previous investigations and highlights new perspectives for investigating these systems.

2 Stochastic and Non-deterministic P Systems: Basic Concepts and Tools

Membrane computing [8] is a branch of natural computing inspired by the hierarchical structure of the living cell. The central model, called *P system*, consists of a membrane structure, the regions of which contain rewriting rules operating on multisets of objects [8]. The P system *evolves* by repeatedly applying rules, mimicking chemical reactions and transportation across membranes or cellular division or death processes, and halts when no more rules can be applied. The most recent developments in this field are reported in [11].

The closeness of this model to the biology makes it highly suited as a specification vehicle for representing biological systems, especially (multi-)cellular systems and molecular interactions taking place in different locations of living cells [12]. Different simple molecular interactions or more complex gene expressions, compartment translocation, as well as cell division and death are specified using multiset rewriting or communication rules, and compartment division or dissolution rules. In the case of stochastic P systems, constants are associated with rules in order to compute their probabilities and time needed to be applied, respectively, according to the Gillespie algorithm [13]. This approach is based on a Monte Carlo algorithm for stochastic simulation of molecular interactions taking place inside a single volume or across multiple compartments.

Definition 1. *A **stochastic P system (SP system)** is a model consisting of a tissue P system with a stochastic semantics [13]:*

$$SP = (O, L, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n) \tag{1}$$

where O is a finite set of objects, called *alphabet,* denoting the entities involved in the system; L is a finite set of *labels* naming compartments; μ is a *membrane structure* composed of $n \geq 1$ membranes defining the regions or compartments of the system and their connections, forming an *arbitrary graph*; $M_i = (l_i, w_i)$, $1 \leq i \leq n$, is the *initial configuration* of the compartment or region defined by the membrane i, where $l_i \in L$ is the label of the compartment and $w_i \in O^*$

is a finite *initial multiset of objects*; $R_i = \{r_1^i, \ldots, r_{m_i}^i\}$, $1 \leq i \leq n$, is a set of *multiset rewriting rules*, of the form: $r_k^i : [\ x \xrightarrow{c_k} y\ ]_{l_i}$, where x and y are multisets of objects (y might be empty) over O, representing the molecular species consumed and produced in the corresponding molecular interaction occurring in the compartment labelled l_i. An application of a rule of this form changes the content of the membrane with label l_i by replacing the multiset x with y. The stochastic constant c_k is used by the Gillespie algorithm [14] in order to compute the probabilities associated with the rules [13].

The model has been used as a basis for a specification language [13, 15] and applied, among others, in unconventional computing using liposomes [2] and specifying a synthetic biology pulse generator [16].

The model also includes communication rules, but these are not discussed in this paper as the system we deal with consists of one single compartment without communication rules. In this case the label of the compartment will be also dropped.

Certain systems can be modelled with P systems which do not require probabilistic features. In [12] some types of P systems without probabilities are presented. These variants are utilised for specifying biological systems. **Kernel P systems (kP systems)** have been introduced as a unifying framework allowing to express within the same formalism many classes of non-deterministic P systems [10, 17]. In this paper we use this class of systems only for very limited purposes, obtaining them directly from the stochastic ones and making use of some tools associated with them. The kP systems derived from SP systems use the same alphabet and rules without kinetic constants. In general, each kP system model has explicitly defined execution strategies for its components. In this paper the execution strategy consists of executing one single rule per step, non-deterministically chosen from the set of rules that can be applied.

The **Infobiotics Workbench** (IBW) tool [16, 15] has been built for modelling and prototyping biological systems exhibiting molecular interactions. It allows to define such systems using the above mentioned formalism, SP systems, providing support for the simulation, verification, analysis and optimisation of these models. The experiments to be discussed later have been performed using IBW. The XOR gate will be modelled using SP systems and then simulated and formally verified. The formal verification is performed with third party tools, PRISM [18] and MC2 [19] (integrated in this framework). The corresponding kP model will be verified using SPIN [20].

PRISM [18] is a very popular and widely used probabilistic model checker. It allows probabilistic properties, supporting PCTL [21] (a probabilistic extension of temporal logic) and Continuous Stochastic Logic (CSL [22]). Both languages make use of special operators to express quantitative information which is useful for a precise, fine grain analysis. The property languages also allow describing *reward*-based properties to express quantitative expressions. PRISM suffers from the same problem exhibited by all model checkers, namely state space explosion

and consequently cannot cope with very large state spaces. This is overcome by an alternative model checking approach, *statistical model checking*.

MC2 [19] is a *statistical* model checker, where properties are analysed against a finite set of simulation traces using statistical methods, e.g. Monte Carlo. Unlike symbolic and numerical methods, e.g. those employed in PRISM, statistical model checkers do not analyse the system *exhaustively*, which increases the performance significantly. In MC2, properties are expressed using PLTL_c [19], a probabilistic extension of LTL with constraints. PLTL_c allows properties with some functions returning *maximum/minimum* values of a species and "*derivative* of the concentration of species at each time point" [19].

The **kP Workbench** (kPW) tool [17] has been built to support kP systems formalism, allowing simulation and formal verification. It uses a specific language, based on kP systems, kP-lingua, allowing to specify non-deterministic rule-based systems. The formal verification is performed using a model checking approach based on SPIN, which is incorporated into the framework. The models written in kP-lingua are automatically translated into SPIN. The non-deterministic version of the XOR gate model will be specified using kP systems and the formal verification will be provided in SPIN.

SPIN [20] is a widely used model checking tool with many applications in concurrent and distributed systems verification. A high level modelling language, PROMELA, suitable for describing concurrent processes and interprocess communication, is at the core of this tool. SPIN provides complete support for *Linear-time Temporal Logic* (LTL) and *on the fly* verification procedures which avoid the necessity to generate the global state space prior to performing a search.

3 XOR Gate and Unconventional Models

In this section we consider a genetic XOR logic gate. This has been designed in various papers, including [5, 6]. The construction used in this paper is taken from [7] where it is defined in GEC, a language for synthetic biology. The gate expresses the green fluorescent protein (GFP) if either of aTc or IPTG molecules are present, but not both. Figure 1 illustrates the genetic construction and the corresponding network.

The XOR device comprises two mechanisms. Each mechanism leads to the production of GFP, when it is activated; but two mechanisms cannot be activated at the same time. Namely, while one is active, the other one is inhibited by some protein.

In this system, the transcription factors LacI and TetR are expressed by a gene controlled by the same promoter. The LacI and TetR proteins work in the opposite way. LacI represses the first mechanism, but promotes the other one. On the other hand, TetR promotes the first mechanism, while inhibiting the second. In other words, both proteins serve as inhibitor and promoter in a complementary fashion. In each mechanism, while one protein is an inhibitor, the other one is promoter. When either of the proteins works as an inhibitor, it binds to the corresponding

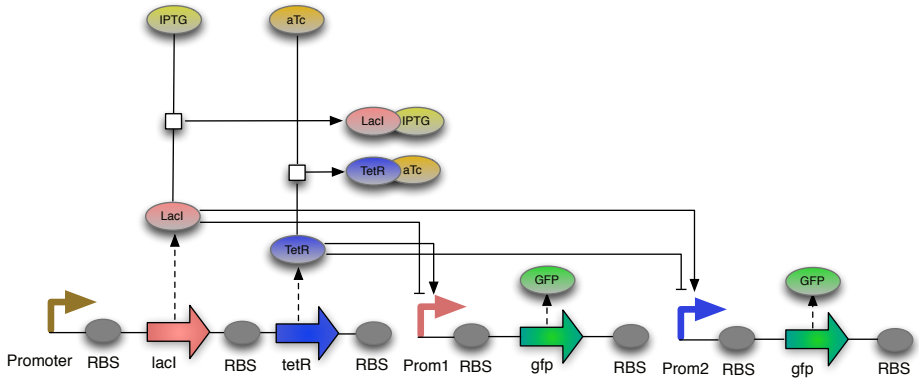


Fig. 1: The genetic parts and design of the XOR gate (redrawn from [7]).

promoter which upregulates the expression of GFP. The XOR device receives two input signals: aTc and IPTG. The aTc and IPTG signals bind to TetR and LacI, respectively, to prevent them interacting with the promoters producing GFP.

Two mechanisms together ensure that the production of GFP will be low when both input signals are set to very low or very high concentrations at the same time. In the former case, LacI and TetR will be produced in abundance, which will then repress the GFP expression. In the latter case, the LacI and TetR concentration will be very low, which is not sufficient to express GFP. On the other hand, if one signal is set to high and the other one is set to low, the device will produce high amount of GFP, hence will act as a Boolean XOR gate.

3.1 Stochastic model

The stochastic model comprises a single compartment with initial concentrations of aTc and IPTG molecules and a set of SP system rules, which govern the kinetic and stochastic behaviour of the system. The initial values are those illustrated in Figures 3. The rewriting rules and the kinetic constants, provided in Table 1, describe the model provided in [7]. A gene controlled by the same promoter expresses LacI and TetR (rules r_1 to r_3). Rules r_{7a} and r_{15a} describe the inhibition of the two mechanisms leading to the GFP production by binding to the promoters that upregulates the production process; r_{7b} and r_{15b} define the debinding process. The activation of the first mechanism by the transcription factor TetR binding to the promoter and the activation of the second mechanism by LacI, are modelled by rules r_{9a}, r_{10}, r_{11} and r_{13a}, r_{14}, r_{17}, respectively. Rules r_{9b} and r_{13b} are debinding reactions. Rules r_4 and r_5 define the binding process involving LacI and IPTG and TetR and aTc, respectively. The degradation process of various molecular species is defined by rules r_{18} to r_{23}.

Table 1: XOR reaction rules.

Rule		Stochastic constant
r_1 :	gene_LacI_TetR $\stackrel{k_1}{\rightarrow}$ gene_LacI_TetR + protein_LacI_TetR	$k_1 = 0.12$
r_2 :	protein_LacI_TetR $\stackrel{k_2}{\rightarrow}$ protein_LacI_TetR + LacI	$k_2 = 0.1$
r_3 :	protein_LacI_TetR $\stackrel{k_3}{\rightarrow}$ protein_LacI_TetR + TetR	$k_3 = 0.1$
r_4 :	LacI + IPTG $\stackrel{k_4}{\rightarrow}$ LacI-IPTG	$k_4 = 1.0$
r_5 :	TetR + aTc $\stackrel{k_5}{\rightarrow}$ TetR-aTc	$k_5 = 1.0$
r_6 :	gene_GFP1 $\stackrel{k_6}{\rightarrow}$ gene_GFP1 + protein_GFP1	$k_6 = 0$
r_{7a} :	gene_GFP1 + LacI $\stackrel{k_{7a}}{\rightarrow}$ gene_GFP1-LacI	$k_{7a} = 1.0$
r_{7b} :	gene_GFP1-LacI $\stackrel{k_{7b}}{\rightarrow}$ gene_GFP1 + LacI	$k_{7b} = 0.01$
r_8 :	gene_GFP1-LacI $\stackrel{k_8}{\rightarrow}$ gene_GFP1-LacI + protein_GFP1	$k_8 = 0$
r_{9a} :	gene_GFP1 + TetR $\stackrel{k_{9a}}{\rightarrow}$ gene_GFP1-TetR	$k_{9a} = 1.0$
r_{9b} :	gene_GFP1-TetR $\stackrel{k_{9b}}{\rightarrow}$ gene_GFP1 + TetR	$k_{9a} = 0.5$
r_{10} :	gene_GFP1-TetR $\stackrel{k_{10}}{\rightarrow}$ gene_GFP1-TetR + protein_GFP1	$k_{10} = 0.1$
r_{11} :	protein_GFP1 $\stackrel{k_{11}}{\rightarrow}$ protein_GFP1 + GFP	$k_{11} = 0.1$
$r_{12:}$	gene_GFP2 $\stackrel{k_12}{\rightarrow}$ gene_GFP2 + protein_GFP2	$k_{12} = 0$
r_{13a} :	gene_GFP2 + LacI $\stackrel{k_{13a}}{\rightarrow}$ gene_GFP2-LacI	$k_{13a} = 1.0$
r_{13b} :	gene_GFP2-LacI $\stackrel{k_{13b}}{\rightarrow}$ gene_GFP2 + LacI	$k_{13b} = 0.5$
r_{14} :	gene_GFP2-LacI $\stackrel{k_{14}}{\rightarrow}$ gene_GFP2-LacI + protein_GFP2	$k_{14} = 0.1$
r_{15a} :	gene_GFP2 + TetR $\stackrel{k_{15a}}{\rightarrow}$ gene_GFP2-TetR	$k_{15a} = 1.0$
r_{15b} :	gene_GFP2-TetR $\stackrel{k_{15b}}{\rightarrow}$ gene_GFP2 + TetR	$k_{15b} = 0.01$
r_{16} :	gene_GFP2-TetR $\stackrel{k_{16}}{\rightarrow}$ gene_GFP2-TetR + protein_GFP2	$k_{16} = 0.0$
r_{17} :	protein_GFP2 $\stackrel{k_{17}}{\rightarrow}$ protein_GFP2 + GFP	$k_{18} = 0.1$
r_{18} :	GFP $\stackrel{k_{18}}{\rightarrow}$	$k_{18} = 0.01$
r_{19} :	LacI $\stackrel{k_{19}}{\rightarrow}$	$k_{19} = 0.01$
r_{20} :	TetR $\stackrel{k_{20}}{\rightarrow}$	$k_{20} = 0.01$
r_{21} :	protein_GFP1 $\stackrel{k_{21}}{\rightarrow}$	$k_{21} = 0.001$
r_{22} :	protein_GFP2 $\stackrel{k_{22}}{\rightarrow}$	$k_{22} = 0.001$
r_{23} :	protein_LacI_TetR $\stackrel{k_{23}}{\rightarrow}$	$k_{23} = 0.001$

Given certain initial values for aTc and IPTG, different output values are obtained for the GFP products, as shown above.

3.2 Non-deterministic model

The rules of the non-deterministic model are obtained directly from the set of SP system rules given in Table 1, by removing the kinetic constants. Some of the rules that do not contribute to the model, with kinetic constants equal to 0 (r_6, r_8, r_{12}, r_{16}), are completely removed. The initial values are kept the same as in the stochastic case. The rules are executed in a non-deterministic manner, as

described earlier in this paper. This non-deterministic model allows to describe all chains of reactions, observe various interactions between species and determine various dependencies between molecules. It will be used in this respect as the basic model for qualitative analysis. As in this case we are interested in an efficient behaviour of the system, some simplifications will be made to the model, whereby the number of molecules will be bounded.

4 Experiments

In this section, we will provide a computational analysis to infer the system dynamics of the genetic XOR gate. This approach complements previous in vitro or in silico implementations of this unconventional computational problem [5, 6, 7] with a set of qualitative and quantitative properties and results. The stochastic model introduced in Section 3.1 and non-deterministic model from Section 3.2 will be used as specifications for the experiments that follow. We note that the complete model and experimental results of the XOR gate can be accessed at[4].

4.1 Non-deterministic Model

The non-deterministic model, discussed in Section 3.2 and obtained from the SP system described in Table 1, will form the basis for translation into Spin.

Model checking results.

The experiments made and reported in this section refer to relationships between species occurring on various reaction pathways. First we verify generic relationships between species. The property

"The GFP is preceded by the production of at least one of LacI or TetR"

is formally expressed as

$$\mathrm{F}\ (\mathrm{GFP} > 0)\ \rightarrow\ \neg((\mathrm{LasR} = 0\ \wedge\ \mathrm{TetR} = 0)\ \mathrm{U}\ \mathrm{GFP} > 0),$$

and the result of this property is true.

We cannot make any direct connections between the signal molecules aTc and IPTG, and the GFP produced, as the system is non-deterministic and any combination of the signal molecules may lead to GFP. However, we can be more specific with respect to the above relationships and refer to the production of a transcription factor and its role as a repressor. More specifically, we verify the property

"When there is no TetR in the system and the LacI represses gene_GFP1 then GFP is produced only by the activation of gene_GFP2"

[4] `http://www.dcs.shef.ac.uk/~konur/models/xor`

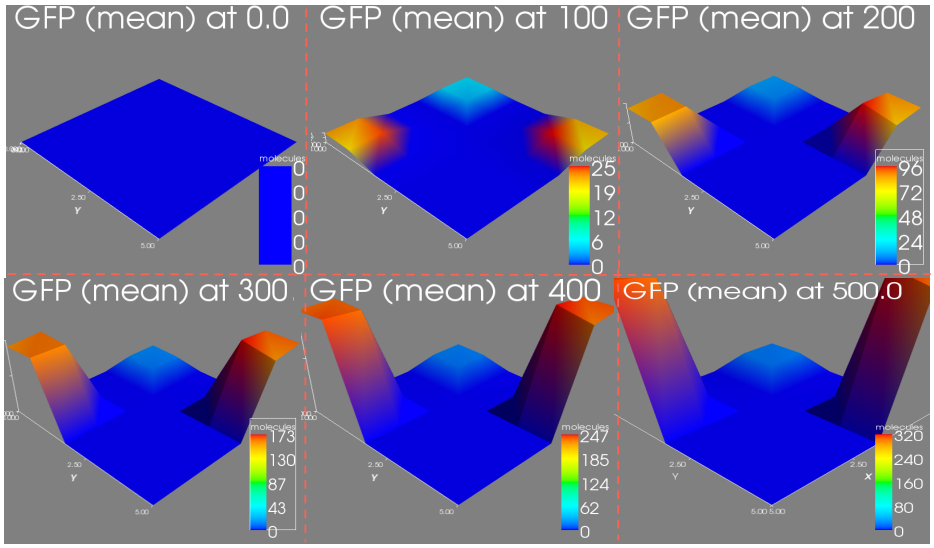


Fig. 2: Simulation of the stochastic model.

which is formally expressed as

$$\mathrm{F}\ (\mathrm{GFP} > 0 \wedge\ \text{gene\_GFP1-LacI} > 0)\ \rightarrow\ \neg((\mathrm{TetR} = 0\ \ \wedge\ \text{gene\_GFP2-LacI} = 0)\ \mathrm{U}\ (\mathrm{GFP} > 0\ \wedge\ \text{gene\_GFP1-LacI} > 0)).$$

The result of this property is true. We can formulate a similar property for TetR.

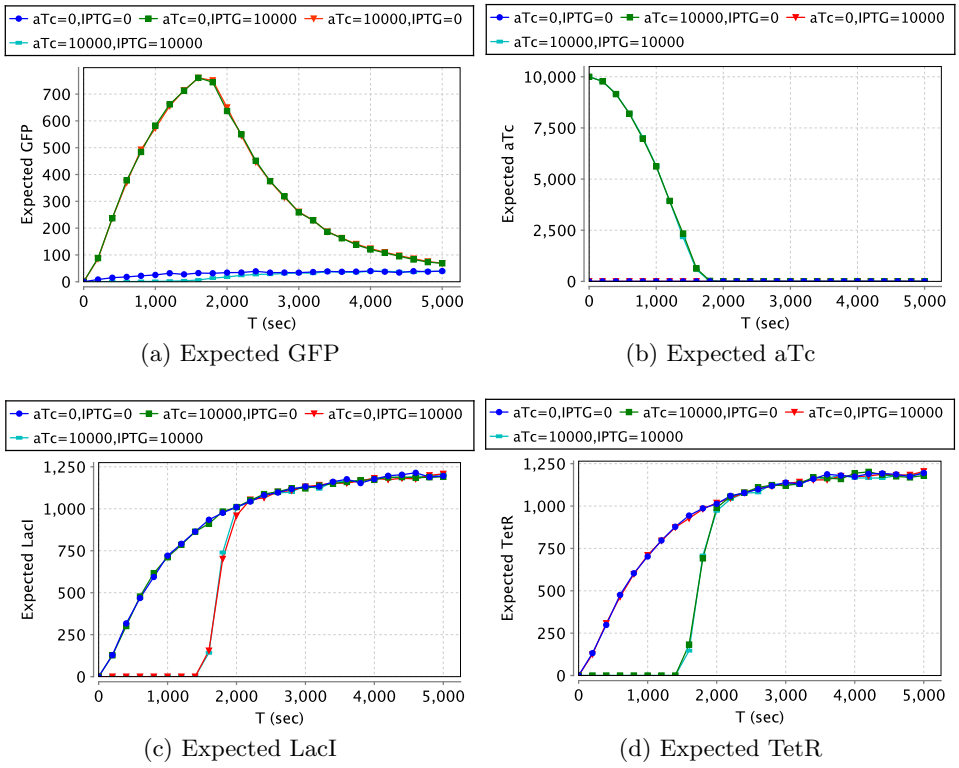
4.2 Stochastic Model

For the stochastic analysis, we have constructed a system model based on SP systems, the modelling language of the Ibw system, using the set of rules discussed in Section 3.1. Below, we summarise some of the experiments that we have carried out using the computational tools integrated into Ibw.

Simulation results.

Figure 2 illustrates the simulation results of the XOR system, performed using the Ibw's Mcss tool, a simulator for multi-compartment SP system models [9]. Ibw provides a GUI to view the simulation results in various formats, e.g. time series, bars, histograms and 3D heat-map animations.

Figure 2 comprises the screen shots of the 3D animation at different time instants. At the top and bottom corners of the lattice both input signals (i.e. aTc



(a) Expected GFP (b) Expected aTc

(c) Expected LacI (d) Expected TetR

Fig. 3: Expected amount of some species based on different initial amounts of aTc and IPTG.

and IPTG) are simultaneously set to very low (i.e. 0) and very high concentrations (i.e. 10000), respectively. Meanwhile, at the left and right corners, one signal is set to very high while the other one is set to very low. As illustrated in the figure, only left and right corners yield a sharp increase in the GFP concentration, ensuring that the designed circuit shows an XOR gate behaviour.

Model checking results.

As discussed above, IBW also permits formal verification of a system using model checking techniques. Since the SP systems allow modelling stochastic models, IBW uses probabilistic model checking tools, currently PRISM and MC2.

Prism results:

We first analyse the amounts of different species over time with four combinations of inputs. The informal property

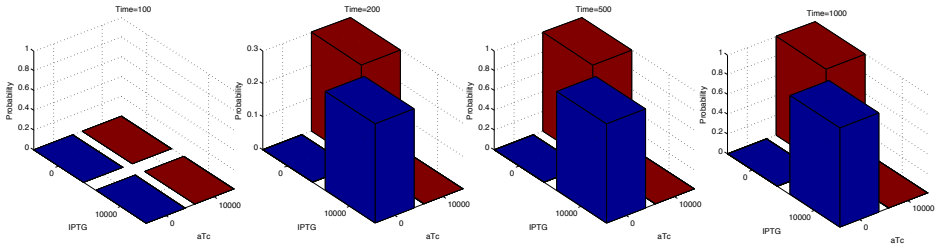


Fig. 4: Probability that GFP exceeds the threshold.

"What is the expected concentration of X at the time instant t?"

is formally expressed as a reward-based formula,

$$\mathrm{R}\{\text{“}X\text{”}\}_{=?}\left[\mathrm{I}=t\right].$$

Figure 3 illustrates the expected amounts of GFP, aTc, LacI and TetR. As shown in Figure 3a, the input combinations aTc=0 – IPTG=10000 and aTc=10000 – IPTG=0 result in a sharp increase in the GFP concentration, whereas the combinations aTc=0 – IPTG=0 and aTc=10000 – IPTG=10000 cause the GFP concentration to stay in low levels, confirming the behaviour of the XOR gate.

Figure 3b shows that if the aTc concentration level is initially set to high, the concentration reduces until it becomes 0. As can be seen in Figure 3d, aTc suppresses the TetR protein by binding to it. After aTc molecules are totally consumed, the TetR concentration starts increasing. We can observe a similar behaviour to Figure 3b, when the IPTG concentration is set to high. IPTG molecules suppress the LacI protein as shown in Figure 3c. These results are inline with the system behaviour, described in Section 3.1.

We now measure the likelihood that the GFP concentration exceeds a certain threshold in any input combination. The property

"What is the probability that GFP *exceeds Thr within t seconds?"*

is formally expressed as

$$\mathrm{P}_{=?}\left[\mathrm{F}^{\leq t}\ \mathrm{GFP} > Thr\right].$$

Figure 4 illustrates the probability values calculated for a threshold value of 100 over different time instants. Clearly, it is almost certainly that GFP exceeds the threshold value for the input combinations aTc=0 – IPTG=10000 and aTc=10000 – IPTG=0. This confirms the desired behaviour.

We now consider a more complex property. Assume that GFP_{ij} (where $i, j \in \{0, 1\}$ represents the state of aTc and IPTG, respectively) denotes the GFP concentration for different input combinations. Namely, if i=0 (resp. j=0), then aTc=0 (resp. IPTG=0), and if i=1 (resp. j=1), then aTc=10000 (resp. IPTG=10000). Then, the property

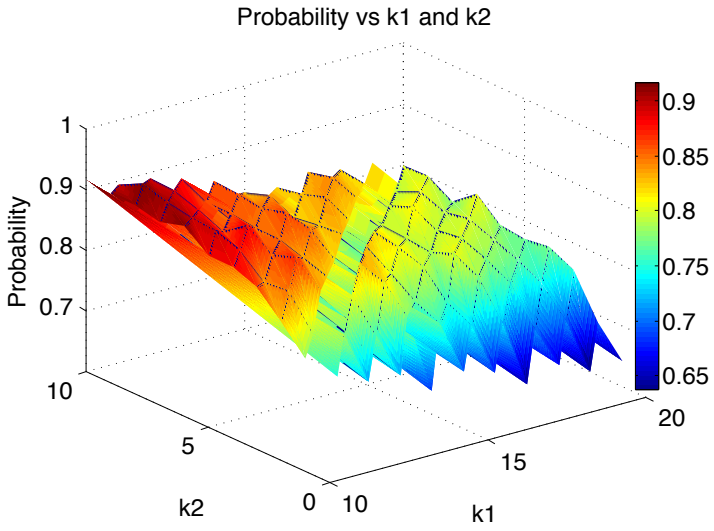


Fig. 5: Parametrised probability formula.

What is the probability that GFP_{01} *and* GFP_{10} *are at least* k_1 *times more than* GFP_{00} *and* GFP_{11}*, and* GFP_{01} *is within the range of* $\mathrm{GFP}_{10} \pm \frac{k_2}{10}$

is formally specified as

$$\begin{aligned} \mathrm{P}_{=?}[\mathrm{F}^{\leq t} \;\; & \mathrm{GFP}_{01} \geq k_1 * \mathrm{GFP}_{00} \;\wedge\; \mathrm{GFP}_{01} \geq k_1 * \mathrm{GFP}_{11} \wedge \\ & \mathrm{GFP}_{10} \geq k_1 * \mathrm{GFP}_{00} \;\wedge\; \mathrm{GFP}_{10} \geq k_1 * \mathrm{GFP}_{11} \wedge \\ & \mathrm{GFP}_{01} \geq (1 - \tfrac{k_2}{10}) * \mathrm{GFP}_{10} \;\wedge\; \mathrm{GFP}_{01} \leq (1 + \tfrac{k_2}{10}) * \mathrm{GFP}_{10}]. \end{aligned}$$

Figure 5 shows the plot based on different k_1 and k_2 values. As expected, the probability becomes higher when k_1 is lower and k_2 is higher, because the formula becomes less strict.

MC2 results:

We now consider a property describing the behaviour in Figure 3a. We want to query

"What is the probability that GFP_{10} *reaches a concentration level of at least* l_1 *times more than the maximum concentrations of* GFP_{00} *and* GFP_{11}*; the concentration then starts decreasing and reduces until it becomes one* l_2 *ration of its maximum level."*

This formula is expressed in PLTL_c as follows:

$$\mathrm{P}_{=?}[\mathrm{F}\ ([\ \mathrm{GFP}_{10}] \geq l_1 * max[\mathrm{GFP}_{00}]\ \wedge\ [\mathrm{GFP}_{10}] \geq l_1 * max[\mathrm{GFP}_{11}] \wedge \\ (\mathrm{F}\ (d[\mathrm{GFP}_{10}] \leq 0\ \wedge\ (\mathrm{F}\ [\mathrm{GFP}_{10}] \leq max[\mathrm{GFP}_{10}]/l_2))))].$$

We have analysed the property for $l_1 = l_2 = 5$, and the probability value returned is 1.0, confirming the behaviour in Figure 3a. We can verify the same property for GFP_{01}, which returns the same result.

5 Conclusions

In this paper we have presented a stochastic P systems model and a non-deterministic one for specifying and studying the behaviour of a genetic XOR gate. These two models are formally analysed using model checking methods revealing qualitative aspects, like expected chain of reactions and dependencies of various species, as well as the quantitative aspects regarding the concentration of certain products with respect to the amount of signal molecules, time to reach certain concentration of molecules or comparisons between maximum concentration achieved for certain species. Our approach is orthogonal to many other unconventional computational investigations or implementations of genetic Boolean gates.

In this line of research, we aim to expand to some other unconventional models, starting with the two genetic XOR gate approaches already mentioned in [5, 6]. We aim also to clarify better the role of various model checkers and types of properties with respect to various systems.

In [23] an interesting prediction and programmability problem for non-DNA molecular self-assembly using porphyrin tiles is investigated. In one of the investigated cases, the self-assembly process is defined as a simple two state probabilistic automaton. The diagonals of a lattice are written with red symbols in one state and blue symbols in the other state, with a small error. This model can be directly represented in Prism and properties regarding the distribution of the two colours on each diagonal or across the lattice are verified. In a forthcoming paper we will be investigating in more details this problem.

Acknowledgements. SK and MG acknowledge EPSRC (EP/I031812/1) support; NK's work is supported by EPSRC (EP/I031642/1, EP/J004111/1, EP/L001489/1). MG and FI are partially supported by CNCS UEFISCDI (PN-II-ID-PCE-2011-3-0688). CD acknowledges an EPSRC studentship.

References

1. Rozenberg, G., Bäck, T., Kok, J.N., eds.: Handbook of Natural Computing. Springer (2012)
2. Smaldon, J., Romero-Campero, F.J., Fernandez Trillo, F., Gheorghe, M., Alexander, C., Krasnogor, N.: A computational study of liposome logic: Towards cellular computing from the bottom up. Systems and Synthetic Biology **4**(3) (2010) 157 – 179

3. Pasparakis, G., Vamvakaki, M., Krasnogor, N., Alexander, C.: Diol-boronic acid complexes integrated by responsive polymers - a route to chemical sensing and logic operations. Soft Matter **4**(20) (2009) 3839 – 3841
4. Chaplin, J.C., Russell, N.A., Krasnogor, N.: Implementing conventional logic unconventionally: Photochromic molecular populations as registers and logic gates. Biosystems **109**(1) (2012) 35 – 51
5. Tamsir, A., Tabor, J.J., Voigt, C.A.: Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. Nature **469**(7329) (2011) 212–215
6. Regot, S., Macia, J., Conde, N., Furukawa, K., Kjellen, J., Peeters, T., Hohmann, S., de Nadal, E., Posas, F., Sole, R.: Distributed biological computation with multicellular engineered networks. Nature **469**(7329) (2011) 207–211
7. Beal, J., Phillips, A., Densmore, D., Cai, Y.: High-level programming languages for biomolecular systems. In: Design and Analysis of Biomolecular Circuits. Springer New York (2011) 225–252
8. Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61**(1) (2000) 108–143
9. Romero-Campero, F.J., Twycross, J., Camara, M., Bennett, M., Gheorghe, M., Krasnogor, N.: Modular assembly of cell systems biology models using P systems. International Journal of Foundations of Computer Science **20**(3) (2009) 427–442
10. Gheorghe, M., Ipate, F., Dragomir, C., Mierlă, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P Systems - Version 1 (2013)
11. Păun, G., Rozenberg, G., Salomaa, A., eds.: The Oxford Handbook of Membrane Computing. Oxford University Press (2009)
12. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J., eds.: Applications of Membrane Computing in Systems and Synthetic Biology. Springer (2014)
13. Romero-Campero, F.J., Twycross, J., Cao, H., Blakes, J., Krasnogor, N.: A multiscale modeling framework based on P systems. In: Membrane Computing. Volume 5391 of LNCS. Springer (2009) 63–77
14. Gillespie, D.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics **22**(4) (1976) 403–434
15. Blakes, J., Twycross, J., Romero-Campero, F.J., Krasnogor, N.: The Infobiotics Workbench: An integrated in silico modelling platform for systems and synthetic biology. Bioinformatics **27**(123) (2011) 3323 – 3324
16. Blakes, J., Twycross, J., Konur, S., Romero-Campero, F.J., Krasnogor, N., Gheorghe, M.: Infobiotics Workbench: A P systems based tool for systems and synthetic biology. In: [12]. Springer (2014) 1–41
17. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierlă, L.: Model Checking Kernel P Systems systems. In: 14th International Conference on Membrane Computing. Volume 8340 of LNCS., Springer (2013) 151–172
18. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Proc. TACAS. Volume 3920 of LNCS. Springer (2006) 441–444
19. Donaldson, R., Gilbert, D.: A Monte Carlo model checker for probabilistic LTL with numerical constraints. Technical report, Bioinformatics Research Centre, University of Glasgow, Glasgow (2008)
20. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering **23**(5) (1997) 275–295

21. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6** (1994) 102–111
22. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time Markov chains. IEEE Transactions on Software Engineering **29**(6) (2003) 524–541
23. Terrazas, G., Lui, L.T., Krasnogor, N.: Spatial computation and algorithmic information content in non-DNA based molecular self-assembly. In: 6th International Workshop on Spatial Computing. (2013) 85–90

P Colony Robot Controller

Miroslav Langer, Luděk Cienciala[1], Lucie Ciencialová[1], Michal Perdek[1], Vladimír Smolka[1]

Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic
{{miroslav.langer, ludek.cienciala, lucie.ciencialova, michal.perdek, vladimir.smolka}@fpf.slu.cz

Summary. P colonies were introduced in 2004 (see [7]) as an abstract computing device composed of independent single membrane agents, reactively acting and evolving in a shared environment. Each agent is equip with set of rules which are structured into simple programs.

We use this very simple symbol processing computational device to build complex robot controllers. Moreover, we group agents into the modules (see [1]). Each module fulfils particular function. This allows us to easily extend our controller or change its function without rebuilding whole P colony. In this paper we introduce simple controller for passing the maze using right-hand rule.

1 Introduction

One of the main tasks of the robotics is to design or program the robot controller. Robot is controlled by controller, which aims to transform outputs from the sensors into the inputs for actuators. The controller can receive input signals from various kinds of sensors, cameras or through communication channels and it has to decide, how this information affect the activity of internal states of the robot and actuators. Robot controller can use different ways to achieve this activity. The controller can be based on the rule-based systems, (fuzzy) expert systems, artificial neural networks and many other techniques.

Recently, we often meet with hybrid controllers, which combine multiple technics to achieve more suitable reaction of robotic systems, or improvement of artificial intelligence. Each of applicable techniques for control has its advantage, but also disadvantage, whether it concerns the complexity, adaptability or scalability etc. We often meet with alternative control methods, which are the result of interdisciplinary convergence of different science disciplines such as biology, genetics, cognitive science, neuroscience, psychology and more.

The concepts of using knowledge of biology can be found in e.g. [4]. Concepts taken from biology such as membrane systems or P-systems will allow us to take

these advantages such as parallelization and distributivity of each parts of the controller.

Controller based on P colonies described in this paper is drawn up as a group of cooperating agents who live in shared environment, through which agents can communicate. Such controller can be used for wide range of tasks associated with control issues.

Throughout the paper we assume that the reader is familiar with the basics of the formal language theory.

2 Preliminaries on the P colonies

P colonies were introduced in 2004 (see [7]) as an abstract computing device composed of independent single membrane agents, reactively acting and evolving in a shared environment. This model is inspired by structure and function of a community of living organisms in a shared environment.

The independent organisms living in a P colony are called agents or cells. Each agent is represented by a collection of objects embedded in a membrane. The number of objects inside each agent is constant during the computation. With each agent is associated a set of simple programs. Each program is composed from the rules which can be of two types. The first type of rules, called the evolution rules, are of the form $a \rightarrow b$. It means that the object a inside the agent is rewritten (evolved) to the object b. The second type of rules, called the communication rules, are of the form $c \leftrightarrow d$. If the communication rule is performed, the object c inside the agent and the object d outside the agent swap their places. Thus after executing the rule, the object d appears inside the agent and the object c is placed outside the agent.

In [6], the set of programs was extended by the checking rules. These rules give the agents an opportunity to opt between two possibilities. The rules are in the form r_1/r_2. If the checking rule is performed, then the rule r_1 has higher priority to be executed over the rule r_2. It means that the agent checks whether the rule r_1 is applicable. If the rule can be executed, then the agent is compulsory to use it. If the rule r_1 cannot be applied, then the agent uses the rule r_2. The program determines the activity of the agent. The agent can change the content of itself or of the environment by programs and it can affect the behaviour of other agents through the environment.

The environment contains several copies of the basic environmental object denoted by e. The environmental object e appears in arbitrary large number of copies in the environment.

This interaction between agents is a key factor in functioning of the P colony. In each moment each object inside the agent is affected by executing the program.

For more information about P systems see [11] or [12].

Definition 1. *The P colony of the capacity k is a construct*
$$\Pi = (A, e, f, V_E, B_1, \ldots, B_n), \text{ where}$$

- *A is an alphabet of the colony, its elements are called objects,*
- *$e \in A$ is the basic object of the colony,*
- *$f \in A$ is the final object of the colony,*
- *V_E is a multiset over $A - \{e\}$, it determines the initial state (content) of the environment,*
- *$B_i,\ 1 \leq i \leq n$, are agents, each agent is a construct $B_i = (O_i, P_i)$, where*
  - *O_i is a multiset over A, it determines the initial state (content) of the agent, $|O_i| = k$,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite multiset of programs, where each program contains exactly k rules, which are in one of the following forms each:*
    - *$a \to b$, called the evolution rule,*
    - *$c \leftrightarrow d$, called the communication rule,*
    - *r_1/r_2, called the checking rule; r_1, r_2 are the evolution rules or the communication rules.*

An initial configuration of the P colony is an $(n+1)$-tuple of strings of objects present in the P colony at the beginning of the computation. It is given by the multiset O_i for $1 \leq i \leq n$ and by the set V_E. Formally, the configuration of the P colony Π is given by $(w_1, \ldots, w_n, w_E)$, where $|w_i| = k,\ 1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent, and $w_E \in (A - \{e\})^\circ$ represents all the objects in the environment different from the object e.

We will use the parallel model of P colonies for the robot controller. That means that each agent tries to find one usable program at each step of the parallel computation. If the number of applicable programs is higher than one, then the agent nondeterministically chooses one of the programs. The maximal possible number of agents is active at each step of the computation.

The configuration is called halting if the set of program labels P satisfying the mentioned conditions above is empty. A set of all possible halting configurations is denoted by H.

Each P colony is characterised by three characteristics; the capacity k, the degree n and the height h; denoted by $NPCOL_{par}K(k, n, h)$. The capacity k is the number of the objects inside each agent, the degree n is the number of agents in the P colony, the height h is the maximal number of programs associated with the agent of the P colony.

2.1 Modularity in the therms of P colonies

In our research of the P colonies we observed that some agents are performing the same function during the computation. In the [1] we grouped agents of the P colony simulating computation of the register machine into the modules. The agents providing subtraction were grouped into the subtraction module, agents providing addition were grouped into the addition module, agents controlling the computation were grouped into the control module, etc. The program of simulated register machine is stored in the control module, so changing the program of the

register machine does not mean reprograming all the agents of the P colony but the change of the control module.

We use this approach to design robot controller. Each individual robot's module like infrared sensors, actuators etc. is represented by the one P colony module. All the P colony modules are driven by the controlling module, which contains controlling logic and drives robot's behaviour.

3 P colony robot controller

P colonies are very simple, but computationally complete string processing computational device working in parallel manner. Data are processed by very primitive computational units using very simple rules formed into the programs. Collaterally working autonomous units sharing common environment provides fast computation device. Dividing agents into the modules allows us to compound agents controlling single robot sensors and actuators. All the modules are controlled by the main controlling unit. Controlling unit collect information from the sensors and send the instructions to the actuators. All the communication is done via the environment.

We construct a P colony with four modules: *Controlling unit*, *Left actuator controller*, *Right actuator controller* and *Infra-red receptor*. Entire colony is amended by the *input* and *output filter*. The input filter codes signals from the robots receptors and spread the coded signal into the environment. In the environment there is the coded signal used by the agents. The output filter decodes the signal from the environment which the actuator controllers sent into it. Decoded signal is forwarded to the robots actuators.

The infra-red receptors consume all the symbols released into the environment by the input filter. It releases actual information from the sensors on demand of the control unit. The infra-red receptors remove unused data from the environment.

The actuator controllers wait for the activating signal from the control unit. After obtaining the activating signal the controllers try to provide demanded action by sending special objects - coded signal for the output filter into the environment. When the action is performed successfully the actuators send the announcement of the successful end of the action to the control unit, the announcement of the unsuccessful end of the action otherwise.

The controlling unit ensures the computation. It controls the behaviour of the robot, it asks the data from the sensors and it sends instructions to the actuators by sending particular symbols to the environment. The controlling unit contains set of programs which provides fulfilling set goal, in this case pass through the maze using the right-hand rule. If the exit from the maze is found; symbol G is appears in the environment, the controlling unit releases into the environment special symbol H, which stops the infra-red receptors and the P colony stops and so the robot.

Let us introduce the formal definition of the mentioned P colony:
$\Pi = (A, e, V_E, (O_1, P_1), \ldots (O_5, P_5), \emptyset)$, where

$A = \{1_L, 1_R, -1_L, -1_R, A_L, A_R, F, F_F, F_O, F_F^F, F_O^F, G, I_F, I_R, H, H_1, H_2, H_3, H_4, L, M_F, R, R_F, R_O, R_F^F, R_O^F, W_i\}$
$V_E = \{e\}$,

Let us describe the meaning of the particular objects:

$1_L, -1_L$ Signal for the output filter - move the left wheel forward/bacward.
$1_R, -1_R$ Signal for the output filter - move the right wheel forward/bacward.
F, L, R Signal from the control unit to the actuator controllers - move forward, turn left/right.
F_F^F, R_F^F Signal from the input filter - no obstacle in front/on the right.
F_O^F, R_O^F Signal from the input filter - obstacle in front/on the right.
F_F, R_F Signal from the IR module to the control unit - no obstacle in front/on the right.
F_O, R_O Signal from the IR module to the control unit - obstacle in front/on the right.
I_F, I_R Signal from the control unit to the IR module - is there an obstacle in front/on the right?
G Maze exit.
H Halting symbol.

Remaining objects are used for inner representation of the actions and as complementary objects.

Particular modules and agents which they contain are defined as follows:

Control Unit:

$O_1 = \{ I_F, I_R, W_i\}$,7
$P_1 = \{ < ee \leftrightarrow A_L A_R; e \leftrightarrow G / e \rightarrow e >; < A_L A_R e \rightarrow I_F I_R W_i >;$
$< I_F I_R \leftrightarrow ee; W_i \rightarrow W_i >;$ $< W_i \rightarrow W_i; ee \leftrightarrow R_F F_F >;$
$< W_i \rightarrow W_i; ee \leftrightarrow R_O F_F >;$ $< W_i \rightarrow W_i; ee \leftrightarrow R_O F_O >;$
$< W_i \rightarrow W_i; ee \leftrightarrow R_F F_O >;$ $< R_F F_O e \rightarrow RRM_F >;$
$< R_O F_F e \rightarrow FFe >;$ $< R_F F_F e \rightarrow RRM_F >;$
$< R_O F_O e \rightarrow LLe >;$ $< FF \leftrightarrow ee; e \rightarrow e >;$
$< LL \leftrightarrow ee; e \rightarrow e >;$ $< RR \leftrightarrow ee; M_F \rightarrow M_F >;$
$< M_F ee \rightarrow FFe >;$ $< A_L A_R G \rightarrow HHH >;$
$< HH \leftrightarrow ee; H \rightarrow H_1 >;$ $< H_1 ee \rightarrow HHH_1 >;$
$< HH \leftrightarrow ee; H_1 \rightarrow H_2 >;$ $< H_2 ee \rightarrow HHH_2 >;$
$< HH \leftrightarrow ee; H_2 \rightarrow H_3 >;$ $< H_3 ee \rightarrow HHH_3 >;$
$< HH \leftrightarrow ee; H_3 \rightarrow H_4 >;$ $< H_4 ee \rightarrow HHH_4 >;$
$< HH \leftrightarrow ee; H_4 \rightarrow e >;$

Control Unit contains program which controls robots behaviour. According to the data obtained from the IR module it sends instructions to the Actuator controllers. It passes the maze using the right-hand rule until it finds symbol G which represents exit from the maze. While it founds the exit the Control unit releases symbol H into the environment which stops the computation.

Left Actuator controller:

$$
\begin{aligned}
O_2 &= \{\, e, e, e\}, \\
P_2 &= \{\, < e{\leftrightarrow}F; e \to 1_L; e \to A_L >; \\
&\quad\ < F{\to}e; 1_L \leftrightarrow e; A_L \leftrightarrow e >; \\
&\quad\ < e{\leftrightarrow}R; e \to 1_L; e \to A_L >; \\
&\quad\ < R{\to}e; 1_L \leftrightarrow e; A_L \leftrightarrow e >; \\
&\quad\ < e{\leftrightarrow}L; e \to -1_L; e \to A_L >; \\
&\quad\ < L{\to}e; -1_L \leftrightarrow e; A_L \leftrightarrow e >;
\end{aligned}
$$

Right Actuator controller:

$$
\begin{aligned}
O_3 &= \{\, e, e, e\}, \\
P_3 &= \{\, < e{\leftrightarrow}F; e \to 1_R; e \to A_R >; \\
&\quad\ < F{\to}e; 1_R \leftrightarrow e; A_R \leftrightarrow e >; \\
&\quad\ < e{\leftrightarrow}R; e \to -1_R; e \to A_R >; \\
&\quad\ < R{\to}e; -1_R \leftrightarrow e; A_R \leftrightarrow e >; \\
&\quad\ < e{\leftrightarrow}L; e \to 1_R; e \to A_R >; \\
&\quad\ < L{\to}e; 1_R \leftrightarrow e; A_R \leftrightarrow e >;
\end{aligned}
$$

Right and left Actuator controllers wait for the activating signal from the Control unit. According to signal they move the robot in required direction by sending appropriate signal to the output filter.

Front Infra-red module:

$$
\begin{aligned}
O_4 &= \{\, e, e, e\}, \\
P_4 &= \{\, < e \leftrightarrow H/e \leftrightarrow F_F^F; ee \to eF_F >; \\
&\quad\ < e \leftrightarrow H/e \leftrightarrow F_O^F; ee \to eF_O >; \\
&\quad\ < F_F \leftrightarrow I_F/F_F \leftrightarrow e; F_F^F e \to ee >; \\
&\quad\ < F_O \leftrightarrow I_F/F_O \leftrightarrow e; F_O^F e \to ee >; \\
&\quad\ < I_F ee \to eee; >\}
\end{aligned}
$$

Right Infra-red module:

$$
\begin{aligned}
O_5 &= \{\, e, e, e\}, \\
P_5 &= \{\, < e \leftrightarrow H/e \leftrightarrow R_F^F; ee \to eR_F >; \\
&\quad\ < e \leftrightarrow H/e \leftrightarrow R_O^F; ee \to eR_O >; \\
&\quad\ < R_F \leftrightarrow I_R/R_F \leftrightarrow e; R_F^F e \to ee >; \\
&\quad\ < R_O \leftrightarrow I_R/R_O \leftrightarrow e; R_O^F e \to ee >; \\
&\quad\ < I_R ee \to eee; >\}
\end{aligned}
$$

IR modules consume all the symbols send by the input filter into the environment. They send actual data to the Control unit on demand.

Robot driven by this P colony is able to pass through simple mazes that are possible to pass using the right-hand rule.

4 Conclusion

We have shown the basic possibilities of controlling the robot using the P colonies and modular approach. We constructed P colony with five simple modules for passing the maze using right-hand rule. With respect to the fact that P colonies are computationally complete devices will the further research be dedicated to the more precise control and fulfilling more difficult and complicated tasks like processing the information from distance sensors.

Remark 1. Article has been made in connection with project IT4Innovations Centre of Excellence, reg. no. CZ.1.05/1.1.00/02.0070.

Research was also supported by the SGS/24/2013 Project of the SU Opava.

References

1. L. Cienciala, L., Ciencialová,, Langer, M.: *Modularity in P colonies with Checking Rules.* In: Revised Selected Papers 12 th International Conference CMC 2011 (Gheorge, M., Păun, Gh., Rozenber, G., Salomaa, A., Verlan, S. eds.), Springer, LNCS 7184, 2012, pp. 104 120.
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, G.: *Cells in environment: P colonies*, Multiple-valued Logic and Soft Computing, 12, 3-4, 2006, pp. 201–215.
3. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: *P colonies with a bounded number of cells and programs.* Pre-Proceedings of the 7^{th} Workshop on Membrane Computing (H. J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), Leiden, The Netherlands, 2006, pp. 311–322.
4. Floreano, D. and Mattiussi, C. (2008). Bio-inspirated Artificial Inteligence: Theories, Methods, and Technologies. MIT Press.
5. Freund, R., Oswald, M.: *P colonies working in the maximally parallel and in the sequential mode.* Pre-Proceedings of the 1^{st} International Workshop on Theory and Application of P Systems (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, pp. 49–56.
6. Kelemen, J., Kelemenová, A.: *On P colonies, a biochemically inspired model of computation.* Proc. of the 6^{th} International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, pp. 40–56.
7. Kelemen, J., Kelemenová, A., Păun, Gh.: *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 at al., eds.) Boston, Mass., 2004, pp. 82–86.
8. Minsky, M. L.: *Computation: Finite and Infinite Machines.* Prentice Hall, Engle-wood Cliffs, NJ, 1967.
9. Păun, Gh.: *Computing with membranes.* Journal of Computer and System Sciences 61, 2000, pp. 108–143.
10. Păun, Gh.: *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
11. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.
12. P systems web page: http://psystems.disco.unimib.it

Constant-Space P Systems with Active Membranes

Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,luca.manzoni,mauri,porreca,zandron}@disco.unimib.it

Summary. We continue the investigation of the computational power of space-constrained P systems. We show that only a constant amount of space is needed in order to simulate a polynomial-space bounded Turing machine. Due to this result, we propose an alternative definition of space complexity for P systems, where the amount of information contained in individual objects and membrane labels is also taken into account. Finally, we prove that, when less than a logarithmic number of membrane labels is available, moving the input objects around the membrane structure without rewriting them is not enough to even distinguish inputs of the same length.

1 Introduction

This paper continues the recent investigations of the computational power of P systems with active membranes by looking at the problems that they are able to solve while working in constant space. It is already known that a super-polynomial amount of space is needed to solve problems outside **PSPACE** [6]. Recently [2], it has been shown that logarithmic space suffices to simulate a polynomial space-bounded Turing machine (TM). Here we show that the constant space is sufficient and, trivially, necessary to solve all problems in **PSPACE**.

This result challenges our intuition about space. How can we even be able to remember, for example, the position of the TM head when we have less than a logarithmic number of bits of information? We discuss the implication of this result and how the current definition of space in P systems can be changed in order to better represent out intuition about "space". With the new definition all the known results involving polynomial (or larger) amount of space, according to the old definition, still hold. Only in the case of P systems with severely tight bounds on space the new definition makes a difference.

Finally, we show a result highlighting the importance of membranes for P systems. In fact, when only movement rules, i.e., send-in, send-out, and dissolution

without rewriting, are allowed on the input symbols and less than a logarithmic number of membrane labels are present, there is no possibility of correctly determining even if two inputs are distinct, unless the ordering of the symbols is discarded.

The paper is organised as follows. The basic notions necessary for the rest of the paper are presented in Section 2. The main result, that is, the simulation of a polynomial space-bounded TM, is described in Section 3. The current definition of space in P systems is discussed and an alternative definition is proposed in Section 4. In Section 5 we examine the limitations arising when only movement but no rewriting is possible for the input symbols. Finally, in Section 6 we present a brief summary of the results and some possible future research directions.

2 Basic Notions

We recall the basic definitions related to P systems with active membranes with input alphabet [7].

Definition 1. *A* P system with (elementary) active membranes *of initial degree* $d \geq 1$ *is a tuple* $\Pi = (\Gamma, \Delta, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$*, where:*

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Δ *is another alphabet, disjoint from* Γ*, called the* input alphabet*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes labelled by elements of* Λ *in a one-to-one way;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are strings over* Γ*, describing the initial multisets of objects placed in the* d *regions of* μ*;*
- R *is a finite set of rules over* $\Gamma \cup \Delta$*.*

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge*, which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

A description of the available kinds of rule follows. This description differs from the original definition [4] only in that new input objects may not be created during the computation.

- *Object evolution rules*, of the form $[a \to w]_h^\alpha$
  They can be applied inside a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w). At most one input object $b \in \Delta$ may appear in w, and only if it also appears on the left-hand side of the rule (i.e., if $b = a$).

- *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$
  They can be applied to a membrane labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β. If $b \in \Delta$ then $a = b$ must hold.
- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β. If $b \in \Delta$ then $a = b$ must hold.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b. If $b \in \Delta$ then $a = b$ must hold.
- *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$
  They can be applied to a membrane labelled by h, having charge α, containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and charges β and γ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes. If $b \in \Delta$ (resp., $c \in \Delta$) then $a = b$ and $c \notin \Delta$ (resp., $a = c$ and $b \notin \Delta$) must hold.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane several evolution rules can be applied simultaneously).
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same principle applies to each membrane that can be involved to communication, dissolution, or elementary division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows.

First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.

- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence of configurations $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules of Π are applicable in $\mathcal{C}_k$. A *non-halting* computation $\mathcal{C} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as language *recognisers* by employing two distinguished objects yes and no; exactly one of these must be sent out from the outermost membrane in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists. Unless otherwise specified, the P systems in this paper are to be considered confluent.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [3].

Definition 2. *Let $\mathcal{E}$, $\mathcal{F}$ be classes of functions over strings. A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* $(\mathcal{E}, \mathcal{F})$-uniform *if the mapping $x \mapsto \Pi_x$ can be described by two functions $F \in \mathcal{F}$ (for "family") and $E \in \mathcal{E}$ (for "encoding") as follows:*

- *$F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n with a distinguished input membrane.*
- *$E(x) = w_x$, where w_x is a multiset encoding the specific input x.*
- *Finally, Π_x is simply Π_n with w_x added to the multiset placed inside its input membrane.*

In particular, a family $\boldsymbol{\Pi}$ is said to be $(\mathbf{L}, \mathbf{L})$*-uniform if the functions E and F can be computed by a deterministic Turing machine in logarithmic space.*

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [3] for further details on the encoding of P systems.

Finally, we describe how space complexity for families of recogniser P systems is measured, and the related complexity classes [5, 7].

Definition 3. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and the total number of objects from Γ (i.e, the non-input objects) they contain. If $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a computation of Π, then the* space required by $\mathcal{C}$ *is defined as*

$$|\mathcal{C}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$$

The space required by Π *itself is then*

$$|\Pi| = \sup\{|\mathcal{C}| : \mathcal{C} \text{ is a computation of } \Pi\}.$$

Finally, let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recogniser P systems, and let $s\colon \mathbb{N} \to \mathbb{N}$. We say that $\boldsymbol{\Pi}$ operates within space bound s *iff $|\Pi_x| \leq s(|x|)$ for each $x \in \Sigma^\star$.*

Definition 4. *We denote by $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(f(n))$ the class of languages decidable by $(\mathbf{L}, \mathbf{L})$-uniform families of* confluent *P systems with active membranes within space bound f. In particular, $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$ denotes the class of languages decidable in constant space.*

3 Simulating Polynomial-Space Turing Machines

The idea behind the simulation of a polynomial-space bounded TM using only constant space in the P system is to use the input objects as a way to store the status of the TM tape. Since by rewriting more than a constant number of input symbols the amount of space would be non-constant, the only way to use them to store the state of the TM tape is to track the position of the symbols inside the membranes. We will use a tape alphabet consisting of two symbols, a and b; the construction presented here immediately generalises to alphabets of any size.

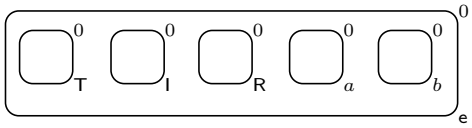
The simulation relies on two main ideas in order to store and retrieve the content of the simulated TM tape. The first idea is that, apart from a short initialisation procedure, the only relevant part of an input object σ_j is its subscript, that is, the position j in the TM tape. We will then have two membranes, named a and b, in which the input objects will be distributed. The interpretation of the fact that an object σ_j is in membrane a is that the j-th cell of the TM contains

the symbol a. Notice that $\sigma = a$ is not required, since this information is not used after the initial phase of the simulation. The second idea is that it is possible to "read" a subscript of an input object σ_j without rewriting it and by using only a constant number of additional objects and membranes. An input object σ_j can use an evolution rule to generate a timer that, after j time steps, changes the charge of a membrane. Any other object that was observing that same membrane, i.e., it was counting together with the timer, is able to obtain the value j of the subscript of σ_j. The two ideas intuitively presented here, to be formalised in the construction below, allow us to store and retrieve the content of the tape of a TM using only a constant number of additional objects and membranes by "moving around" the input objects.

The simulation is divided into three phases. The first one is the initialisation, in which the input objects are distributed across the membranes of the systems. The second phase is simulation of a step of the TM, which requires the third phase, where the P system is reset in order to simulate the next step.

3.1 Membrane structure

The membrane structure consists of six membranes:



Each of these membranes has a specific semantics:

- T is a "temporary storage". It will contain objects that are not needed in that particular stage of the computation, and, then it will be emptied and the object contained will be moved to one of the other membranes. It also serves as the input membrane.
- I is used only during the initialisation, acting as a container for the "dispatching machinery" that moves an input object to the correct membrane.
- R is a membrane used to check the subscript of a specific input symbol. The input object σ_j generates a timer that changes the charge of the membrane after j time steps.
- Membrane a (resp. b) contains all input objects σ_j such that, in the currently simulated step of the TM, the j-th cell of the tape contains the symbol a (resp. b).
- e is the external membrane, containing all the others.

If the tape alphabet contains more than two symbols, it is possible to add more membranes inside e, one for each symbol.

3.2 Initialisation

The initial configuration of the P system contains all the input objects in membrane T and one auxiliary object move in membrane e. The initialisation procedure moves each input object a_j (resp., b_j) to membrane a (resp., b) and generates an object $q_{0,a}$, where q is the initial state of the TM, 0 indicates the position of the TM on the tape, and a indicates the fact that the P system simulating the TM will check if the symbol on the TM tape at position 0 is a. An example of the movement of one object to the correct membrane is shown in Fig. 1.

In the description of the simulation we assume that n is the size of the input and $p(n)$ is the length of the tape of the TM, which is polynomial in n.

The following set of rules uses the object move to transport an input object outside of membrane T or, if the membrane is empty, to start the simulation of the first step of the TM.

$$
\begin{aligned}
&\mathsf{move}\ [\ ]^0_{\mathsf{T}} \to [\mathsf{move}]^-_{\mathsf{T}} && \\
&[\mathsf{move} \to \mathsf{move}']^-_{\mathsf{T}} && \\
&[\mathsf{move}']^0_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ \mathsf{open} && \\
&[a_i]^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ a_i && 0 \le i < p(n) \\
&[b_i]^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ b_i && 0 \le i < p(n) \\
&[\mathsf{move}']^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ q_{0,a} &&
\end{aligned}
$$

The next set of rules is used to move an input object from membrane e to I:

$$
\begin{aligned}
&\mathsf{open}\ [\ ]^0_{\mathsf{I}} \to [\mathsf{open}]^+_{\mathsf{I}} && \\
&[\mathsf{open}]^0_{\mathsf{I}} \to [\ ]^-_{\mathsf{I}}\ \mathsf{wait} && \\
&a_i\ [\ ]^+_{\mathsf{I}} \to [a_i]^0_{\mathsf{I}} && 0 \le i < p(n) \\
&b_i\ [\ ]^+_{\mathsf{I}} \to [b_i]^0_{\mathsf{I}} && 0 \le i < p(n)
\end{aligned}
$$

The following rules move an input object a_i (resp., b_i) to membrane a (resp., b):

$$
\begin{aligned}
&[a_i \to a_i\ \mathsf{goto}_a]^0_{\mathsf{I}} && 0 \le i < p(n) && (1) \\
&[b_i \to a_i\ \mathsf{goto}_b]^0_{\mathsf{I}} && 0 \le i < p(n) && (2) \\
&[\mathsf{goto}_a]^0_{\mathsf{I}} \to [\ ]^-_{\mathsf{I}}\ \mathsf{goto}_a && && \\
&[\mathsf{goto}_b]^0_{\mathsf{I}} \to [\ ]^-_{\mathsf{I}}\ \mathsf{goto}_b && && \\
&[a_i]^-_{\mathsf{I}} \to [\ ]^0_{\mathsf{I}}\ a_i && 0 \le i < p(n) && \\
&[b_i]^-_{\mathsf{I}} \to [\ ]^0_{\mathsf{I}}\ b_i && 0 \le i < p(n) && \\
&\mathsf{goto}_a\ [\ ]^0_a \to [\mathsf{goto}_a]^+_a && && \\
&\mathsf{goto}_b\ [\ ]^0_b \to [\mathsf{goto}_b]^+_b && && \\
&[\mathsf{goto}_a \to \epsilon]^+_a && &&
\end{aligned}
$$

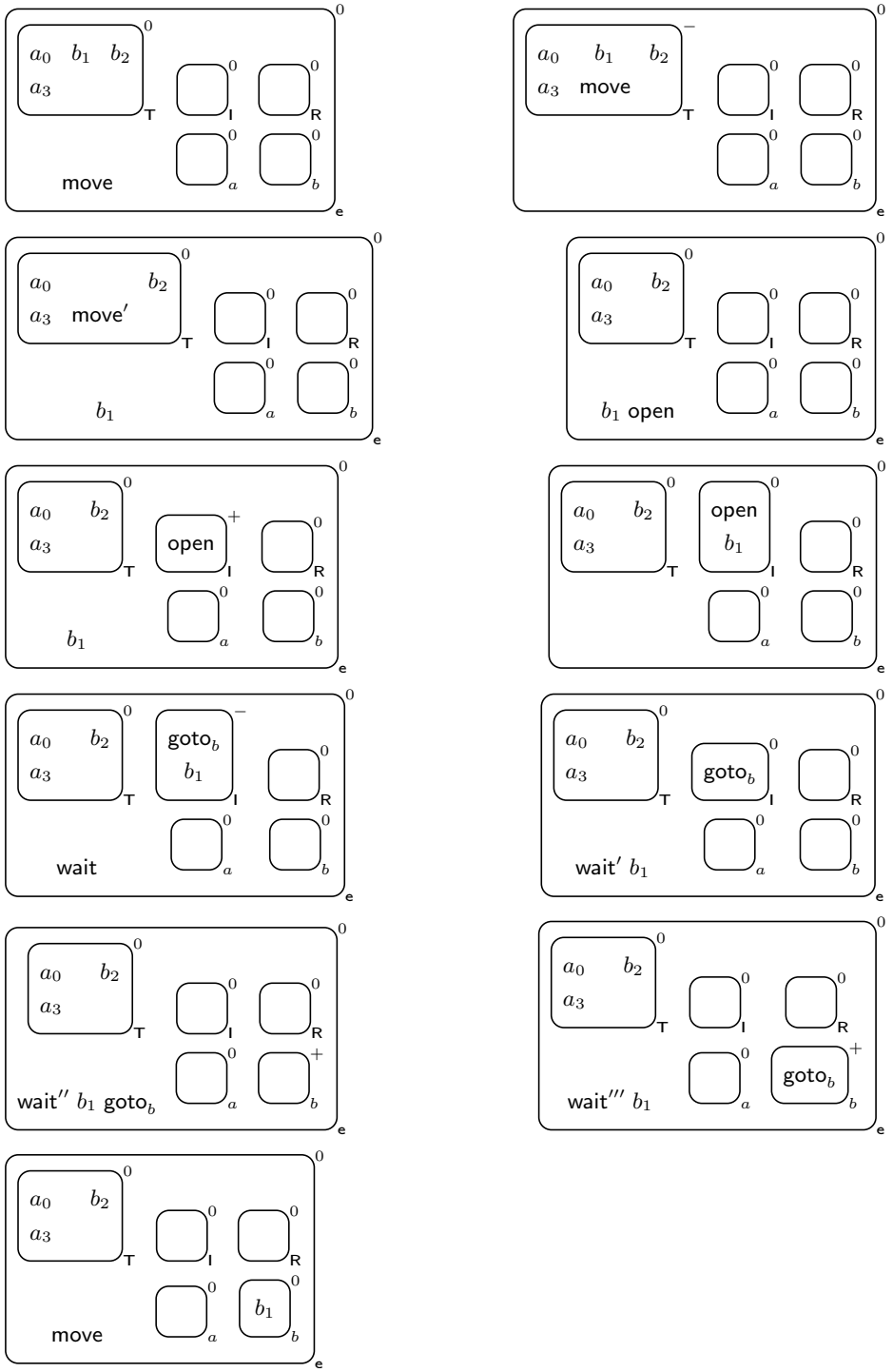


Fig. 1. The movement of an input symbol to the correct membrane during the initialisation phase.

$$
\begin{aligned}
&[\mathsf{goto}_b \to \epsilon]^+_b \\
&a_i\ [\ ]^+_a \to [a_i]^0_a && 0 \le i < p(n) \\
&b_i\ [\ ]^+_b \to [b_i]^0_b && 0 \le i < p(n)
\end{aligned} \tag{3}
$$

Finally, the auxiliary object in membrane e has to wait for three steps before moving another input object outside of membrane T:

$$
\begin{aligned}
&[\mathsf{wait} \to \mathsf{wait}']^0_{\mathsf{e}} \\
&[\mathsf{wait}' \to \mathsf{wait}'']^0_{\mathsf{e}} \\
&[\mathsf{wait}'' \to \mathsf{wait}''']^0_{\mathsf{e}} \\
&[\mathsf{wait}''' \to \mathsf{move}]^0_{\mathsf{e}}
\end{aligned}
$$

After the initialisation phase, all the input objects of the form a_i (resp., b_i) are located in membrane a (resp., b), and the actual simulation of the TM can start.

3.3 Simulation of a Step of the Turing Machine

To simulate a step of the TM we first define how its configuration is encoded as a configuration of the P system. Let $c_0, c_1, \ldots, c_{p(n)-1}$, with $c_i \in \{a, b\}$, be the tape of the TM at the current time step, and $\sigma_0, \ldots, \sigma_{n-1}$ the initial content of the tape. Then the P system contains, in membrane a, all σ_j such that $c_j = a$ and, in membrane b, all the input objects σ_j such that $c_j = b$. Notice that this allows a complete reconstruction of the tape of the TM. The state q of the TM and the position i of the head are encoded in an object $q_{i,\tau}$ in membrane e, where $\tau \in \{a, b\}$ indicates that the P system will check if the symbol in position i of the tape is τ.

The simulation proceeds as follows:

- The object $q_{i,\tau}$ moves an object σ_j from membrane τ, i.e., either a or b, to membrane R.
- In membrane R the object σ_j produces a timer that counts from j to 0, while $q_{i,\tau}$ counts from i to 0. When the first timer stops, it changes the charge of R, that is immediately changed again by the object σ_j exiting from R. This makes it possible to determine if $i = j$. In that case, the symbol on the tape of the TM in position i is actually τ, and it is possible to perform a step of the simulated machine. In the other case, i.e., $i \neq j$, the object σ_j is moved into membrane T and we return to the previous step.
- When membrane τ is empty, or the correct input object was found in the previous step, it is necessary to move back the objects from T to membrane τ. The search for the input object having subscript i will then proceed in membrane b (when $\tau = a$) or a (when $\tau = b$).

We will now detail the rules necessary to formally define the previous algorithm. In order to shorten the notation, in the following description we are going to write only half of the rules, those involving a. The missing half is obtained by swapping a and b.

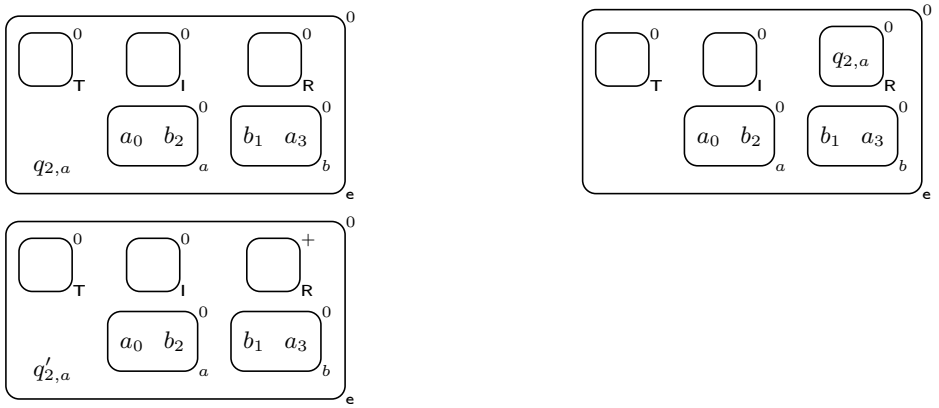


Fig. 2. The symbol $q_{2,a}$ changes the charge of the membrane R before starting the next phase of the simulation (continues in Fig 3).

Reading the Symbol under the Tape Head

This first set of rules uses the object $q_{i,a}$ to change the charge of membrane R to +, as shown in Fig. 2. The set Q' denotes the non-finals states of Q.

$$\begin{aligned} q_{i,a}\ [\ ]^0_{\mathsf{R}} &\to [q_{i,a}]^0_{\mathsf{R}} && \text{for } q \in Q',\ 0 \le i < p(n) \\ [q_{i,a}]^0_{\mathsf{R}} &\to [\ ]^+_{\mathsf{R}}\ q'_{i,a} && \text{for } q \in Q',\ 0 \le i < p(n) \end{aligned}$$

The following rules move an input object from membrane a to membrane R.

$$\begin{aligned} q'_{i,a}\ [\ ]^0_a &\to [q'_{i,a}]^-_a && \text{for } q \in Q',\ 0 \le i < p(n) \\ [q'_{i,a} &\to q''_{i,a}]^-_a && \text{for } q \in Q',\ 0 \le i < p(n) \\ [\sigma_j]^-_a &\to [\ ]^0_a\ \sigma_j && \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \\ [q''_{i,a}]^0_a &\to [\ ]^0_a\ q_{i,a,\mathsf{read}} && \text{for } q \in Q',\ 0 \le i < p(n) \\ [q''_{i,a}]^-_a &\to [\ ]^0_a\ q_{i,a,\mathsf{reset}} && \text{for } q \in Q',\ 0 \le i < p(n) \\ \sigma_j\ [\ ]^+_{\mathsf{R}} &\to [\sigma_j]^0_{\mathsf{R}} && \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \end{aligned} \tag{4}$$

The next rules allow us to compare the position of the head of the TM (stored as a subscript of the object $q_{i,a,\mathsf{read}}$) with the subscript j of the object σ_j that has left membrane a. To accomplish this, the object σ_j in membrane R produces the object timer_j. At the same time step in which timer_j is produced, the object $q_{i,a,\mathsf{read}}$ enters R, changing its charge to +. Both timer_j and $q_{i,a,\mathsf{read}}$ rewrite themselves in order to count from j (resp., i) to 0. In this way, it is possible to determine if $i = j$. If it is so, then $q_{i,a,\mathsf{found}}$ exits from R. If not, it is $q_{i,a,\mathsf{not\text{-}found}}$ that appears. An example of application of those rules is presented in Fig. 3.

$$[\sigma_j \to \sigma_j\ \mathsf{timer}_j]^0_{\mathsf{R}} \qquad \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \tag{5}$$

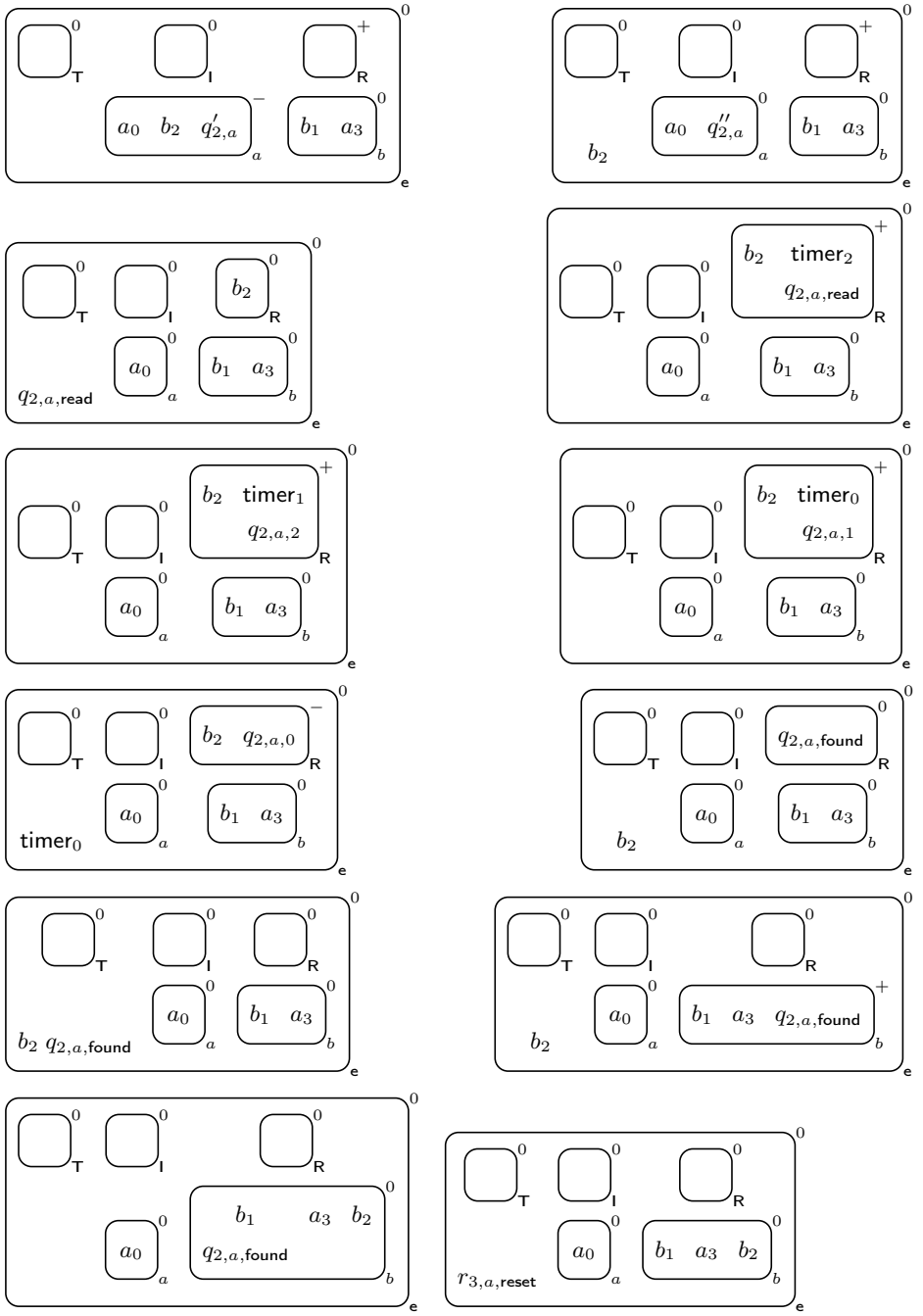


Fig. 3. The symbol $q_{2,a}$ is used to discover that position 2 of the tape contains a. After that, a transition to state r and a movement of the tape head to position 3 is performed.

$$
\begin{array}{ll}
q_{i,a,\mathsf{read}}\ [\ ]^0_{\mathsf{R}} \to [q_{i,a,\mathsf{read}}]^+_{\mathsf{R}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[\mathsf{timer}_j \to \mathsf{timer}_{j-1}]^+_{\mathsf{R}} & \text{for } 1 \le j \le p(n) \\
[q_{i,a,\mathsf{read}} \to q_{i,a,i}]^+_{\mathsf{R}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q_{i,a,k} \to q_{i,a,k-1}]^+_{\mathsf{R}} & \text{for } q \in Q',\ 0 \le i < p(n),\ 1 \le k \le p(n) \\
[\mathsf{timer}_0]^+_{\mathsf{R}} \to [\ ]^-_{\mathsf{R}}\ \mathsf{timer}_0 & \\
[\mathsf{timer}_0 \to \epsilon]^0_{\mathsf{e}} & \\
[\sigma_j]^-_{\mathsf{R}} \to [\ ]^0_{\mathsf{R}}\ \sigma_j & \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \\
[q_{i,a,0} \to q_{i,a,\mathsf{found}}]^-_{\mathsf{R}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q_{i,a,0} \to q_{i,a,\mathsf{not\text{-}found}}]^+_{\mathsf{R}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q_{i,a,0} \to q_{i,a,\mathsf{not\text{-}found}}]^0_{\mathsf{R}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q_{i,a,\mathsf{found}}]^0_{\mathsf{R}} \to [\ ]^0_{\mathsf{R}}\ q_{i,a,\mathsf{found}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q_{i,a,\mathsf{not\text{-}found}}]^0_{\mathsf{R}} \to [\ ]^0_{\mathsf{R}}\ q_{i,a,\mathsf{not\text{-}found}} & \text{for } q \in Q',\ 0 \le i < p(n)
\end{array} \tag{6}
$$

If $i \neq j$ we need to move the selected input object σ_j to membrane T:

$$
\begin{array}{ll}
q_{i,a,\mathsf{not\text{-}found}}\ [\ ]^0_{\mathsf{T}} \to [q_{i,a,\mathsf{not\text{-}found}}]^+_{\mathsf{T}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
\sigma_j\ [\ ]^+_{\mathsf{T}} \to [\sigma_j]^0_{\mathsf{T}} & \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \\
[q_{i,a,\mathsf{not\text{-}found}}]^0_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ q_{i,a} & \text{for } q \in Q',\ 0 \le i < p(n)
\end{array}
$$

On the other hand, if $i = j$ we have correctly identified the symbol under the head of the TM. Assume $\delta(q,a) = (r,\tau,d)$ with r non final; then we define the following rules:

$$
\begin{array}{ll}
q_{i,a,\mathsf{found}}\ [\ ]^0_{\tau} \to [q_{i,a,\mathsf{found}}]^+_{\tau} & \text{for } 0 \le i < p(n) \\
\sigma_i\ [\ ]^+_{\tau} \to [\sigma_i]^0_{\tau} & \text{for } \sigma \in \{a,b\},\ 0 \le i < p(n) \\
[q_{i,a,\mathsf{found}}]^0_{\tau} \to [\ ]^0_{\tau}\ r_{i+d,a,\mathsf{reset}} & \text{for } 0 \le i < p(n)
\end{array} \tag{7}
$$

Notice that, in the case of a nondeterministic TM, the simple repetition of rule (7) for each $(r,\tau,d) \in \delta(q,a)$ assures a non-deterministic simulation on a non-confluent P system.

Notice that the last rule has a change in both the state and in the position of the head. The presence of reset in the subscript indicates that this object will move all the objects from membrane T to membrane a. This happens also in another case (see rule (4)) but without any change of state and position of the TM head. This is an intended behaviour, since in both cases, before continuing the simulation, we need to restore the configuration of the P system by emptying membrane T. After that, the simulation with proceed with the object $r_{i+d,b}$ in this case, and $q_{i,b}$ in the other, as intended.

Clean-up

The following set of rules use the object $q_{i,a,\mathsf{reset}}$ to move all the objects from membrane T to membrane a. After that, the object is rewritten into $q_{i,b}$.

$$
\begin{array}{ll}
q_{i,a,\mathsf{reset}}\ [\ ]^0_{\mathsf{T}} \to [q_{i,a,\mathsf{reset}}]^-_{\mathsf{T}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[\sigma_j]^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ \sigma_j & \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \\
[q_{i,a,\mathsf{reset}} \to q'_{i,a,\mathsf{reset}}]^-_{\mathsf{T}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q'_{i,a,\mathsf{reset}}]^0_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ q'_{i,a,\mathsf{reset}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
q'_{i,a,\mathsf{reset}}\ [\ ]^0_a \to [q'_{i,a,\mathsf{reset}}]^+_a & \text{for } q \in Q',\ 0 \le i < p(n) \\
\sigma_j\ [\ ]^+_a \to [\sigma_j]^0_a & \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \\
[q'_{i,a,\mathsf{reset}}]^0_a \to [\ ]^0_a\ q_{i,a,\mathsf{reset}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q'_{i,a,\mathsf{reset}}]^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ q_{i,b} & \text{for } q \in Q',\ 0 \le i < p(n)
\end{array}
$$

Now all the input objects have been moved to either membrane a or membrane b, and the P system proceeds by checking whether the tape head is reading the symbol b in state q.

3.4 Halting

If $\delta(q,a) = (r,\tau,d)$ and r is an accepting (resp., rejecting) state of the TM, then, when simulating the last transition, instead of rule (7) one of the following rules is applied:

$$
\begin{array}{ll}
[q_{i,a,\mathsf{found}}]^0_\tau \to [\ ]^0_\tau\ \mathsf{yes} & \text{for } 0 \le i < p(n), \text{ if } r \text{ is accepting} \\
[q_{i,a,\mathsf{found}}]^0_\tau \to [\ ]^0_\tau\ \mathsf{no} & \text{for } 0 \le i < p(n), \text{ if } r \text{ is rejecting}
\end{array}
$$

Finally, the object yes or no is sent out from the outermost membrane, as the last computation step by, via the rules

$$
\begin{array}{l}
[\mathsf{yes}]^0_{\mathsf{e}} \to [\ ]^0_{\mathsf{e}}\ \mathsf{yes} \\
[\mathsf{no}]^0_{\mathsf{e}} \to [\ ]^0_{\mathsf{e}}\ \mathsf{no}
\end{array}
$$

No further rule can then be applied, since the only remaining objects are the input objects located in membranes T, a, and b, where they are not subject to any rule, as the membrane charges are neutral.

3.5 Main Result

The simulation of one step of the TM requires at most a polynomial number of steps of the P system, since each of the different "phases" (initialisation, checking if the input symbol σ_j has a subscript corresponding to the current position of

the head of the TM, and moving the objects from membrane T to either a or b) requires at most polynomial time – actually $O(p(n))$ – and it is repeated at most once for each cell of the tape, that is, at most $p(n)$ times.

Even if we have presented a simulation of a TM having only two tape symbols, it is possible to extend the simulation to arbitrary alphabets by using one membrane for each symbol, similarly to membranes a and b, and by adding the corresponding rules. Therefore, in the following theorem we assume that a blank tape symbol $\sqcup$ is available.

Theorem 1. $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1)) = \mathbf{PSPACE}$.

Proof. Let $L \in \mathbf{PSPACE}$, and let M be a TM deciding L in space $p(n)$). We can construct a family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ such that $L(\boldsymbol{\Pi}) = L$ by letting $F(1^n) = \Pi_n$, where Π_n is the P system simulating M on inputs of length n, and $E(x_0 \cdots x_{n-1}) = x_{1,1} \cdots x_{n-1,n-1} \sqcup_n \cdots \sqcup_{p(n)-1}$, i.e., by padding the input string x with $p(n) - n$ blank symbols before indexing the result with the positions of the symbols. Both F and E can be computed in logarithmic space by Turing machines, since they only require adding subscripts having a logarithmic number of bits to rules or strings having a fixed structure, and the membrane structure is fixed for all Π_n. Since the simulation of M only requires a constant number of membranes and non-input objects, the inclusion $\mathbf{PSPACE} \subseteq (\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$ follows. The reverse inclusion was proved in [6]. □

4 Rethinking the Definition of Space

The result of Theorem 1 shows that constant-space P systems with active membranes have the same computational power of Turing machines working in polynomial space. This raises some doubts about the definition of space complexity for P systems adopted until now [5]: does counting each non-input object and each membrane as unitary space really capture an intuitive notion of "space"?

From an information-theoretic perspective, we may observe that the constant number of non-input objects employed by the simulation of Section 3.5 actually encode $\Theta(\log n)$ bits of information, since they are taken from an alphabet Γ of polynomial size. It may be argued that this amount of information needs a proportional amount of physical storage space (e.g., as DNA molecules of comparable size). Similarly, the membranes themselves, being identified by a label, contain $\Theta(\log |\Lambda|)$ bits of information, which must have a physical counterpart.[1]

A more accurate estimate of the space required by a configuration of a P system might be given by the following alternative definition:

[1] We refer to the objects and membrane labels actually appearing during the course of the computation here; if part of the alphabet Γ or some labels from Λ never appear in a configuration, then the information content might be smaller.

Definition 5. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the number of membranes in the current membrane structure* multiplied by $\log|\Lambda|$, *plus the total number of objects from Γ (i.e, the non-input objects) they contain* multiplied by $\log|\Gamma|$.

Adopting this stricter definition does not significantly change space complexity results for polynomial or larger upper bounds, i.e., the complexity classes $\mathbf{PMCSPACE}_{\mathcal{AM}}$, $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$, and larger ones [1] remain unchanged.

On the other hand, the simulation described in this paper would require logarithmic space according to Definition 5. Furthermore, the space bounds of the previous simulation of polynomial-space Turing machines by means of logarithmic-space P systems with active membranes [2] also increase to $\Theta(\log n \log\log n)$, since in that case each configuration of the P systems contained $\Theta(\log n)$ membranes with distinct labels and $O(1)$ non-input objects.

It remains to be established if space (as in Definition 5) can be freely exchanged between objects and labels, or if one of the two is strictly more powerful.

5 Computing without Evolving Input Objects

In this section we are going to show that, if input object are only moved around the membrane structure (without being themselves rewritten into other objects), evolution rules involving the input objects, such as (1), (2), and (5), are essential in order to perform a simulation of a TM. In fact, if only non-rewriting send-in, send-out, and dissolution rules are applied to input symbols, and the number of membrane labels is $o(\log n)$, it is impossible even to correctly distinguish two input strings of the same length. This happens independently of the space used by the P systems, as long as the multiset encoding of the input $x \in \Sigma^\star$ is "simple":

Definition 6. *Let A be an alphabet containing Σ, and let*

$$s\colon A^\star \to \{\sigma_i : \sigma \in A, i \in \mathbb{N}\}^\star$$

be the function defined by $s(x_0 \cdots x_{n-1}) = x_{0,0} \cdots x_{n-1,n-1}$, i.e., the function subscripting each symbol with its position in the string.

We say that an encoding E of $\Sigma^\star$ is "simple" if there exists a function $g\colon \mathbb{N} \to A^\star$ such that $E(x) = s(x \cdot g(|x|))$, i.e., $E(x)$ is the original input string x, concatenated with a string depending only on the length of x, and indexed with the positions of its symbols.

Notice that the encoding employed in Theorem 1 is indeed simple.

When the encoding is simple and the input alphabet is at least binary, P systems with the limitations described above accepting (resp., rejecting) a long enough string x also accept (resp., reject) another string obtained by swapping two symbols of x.

Theorem 2. *Let $\boldsymbol{\Pi}$ be a family of $(\mathcal{E}, \mathcal{F})$-uniform, possibly non-confluent P systems with active membranes, where $\mathcal{F}$ is unrestricted, and $\mathcal{E}$ is the class of simple encodings. Suppose that the only rules involving input symbols are send-in communication, send-out communication and membrane dissolution, that these rules never rewrite the input symbol, and that the family uses $o(\log n)$ membrane labels.*

Then, there exists $n_0 \in \mathbb{N}$ such that, for each string $x = x_0 \cdots x_{n-1} \in \Sigma^\star$ with $|x| \geq n_0$, there exist $i < j < n$ such that x can be written as $u \cdot x_i \cdot v \cdot x_j \cdot w$ and $x \in L(\boldsymbol{\Pi})$ if and only if $u \cdot x_j \cdot v \cdot x_i \cdot w \in L(\boldsymbol{\Pi})$.

Proof. Let $\boldsymbol{\Pi}$ be a family of P systems as defined in the statement of the theorem, let $F \in \mathcal{F}$ be its "family" function, and let

$$F(1^n) = \Pi_n = (\Gamma, \Delta, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R).$$

Assume that the objects in Δ have, in R, only rules of type send-in, send-out, and dissolution not rewriting them. We impose no restriction on rules involving objects in Γ. Let us consider the possible rules applicable to a fixed object $a \in \Delta$ and respecting the imposed restrictions:

- There are at most $3^2|\Lambda|$ send-in rules of the form $a\ [\ ]_h^\alpha \to [a]_h^\beta$, since it is possible to choose the label of the membrane and the two charges.
- Similarly, there are at most $3^2|\Lambda|$ send-out rules of the form $[a]_h^\alpha \to [\ ]_h^\beta\ a$.
- The number of dissolution rules of the form $[a]_h^\alpha \to a$ is at most $3|\Lambda|$, since, contrarily to the last two cases, there is no charge on the right-hand side of the rule.

Thus, there are $21|\Lambda|$ possible rules per input object and $2^{21|\Lambda|}$ possible sets of rules involving each input object. Since the encoding of the input string is simple, each input object has the form σ_i for some $\sigma \in A$; hence, for each position i in the input multiset there are $\left(2^{21|\Lambda|}\right)^{|A|} = 2^{21|\Lambda||A|}$ possible sets of rules.

A necessary condition to distinguish two input objects $a, b \in \Delta$ is that the set of rules involving b cannot be simply obtained by replacing a with b in the set of rules involving a (i.e., their sets of rules are not isomorphic); otherwise, replacing a with b in the input multiset would not change the result of the computation of Π. In particular, this holds for the first n input objects $x_{0,0}, \ldots, x_{n-1,n-1}$, obtained by indexing $x = x_0 \cdots x_{n-1} \in \Sigma^n$. In order to be able to distinguish these n input objects it is necessary that

$$2^{21|\Lambda||\Sigma|} \geq n$$

that is, that the sets of rules associated with the first n objects are pairwise non isomorphic. This means that

$$|\Lambda| \geq \frac{\log n}{21|\Sigma|}$$

But $|\Lambda|$ is $o(\log n)$; hence, the inequality does not hold for large enough n. Instead, there exists n_0 such that, for each $n \geq n_0$, there are two indistinguishable positions i and j with $0 \leq i < j < n$: for each $x = u \cdot x_i \cdot v \cdot x_j \cdot w \in \Sigma^n$, either x and $u \cdot x_j \cdot v \cdot x_i \cdot w$ are both accepted, or they are both rejected. □

6 Final Remarks

In this paper we have solved the problem of determining the computational power of P systems working in constant space, by showing that they can simulate polynomial-space bounded Turing machines. The simulation is also efficient, in the sense that it is only polynomially slower than the original machine.

The solution of this problem raised some interesting questions about the ability of the current definition of space to capture our intuitions about the size of P systems. We have challenged the existing definition by considering also the number of bits necessary to encode the auxiliary objects and the labels of the membranes. While the new definition does not change any result involving an amount of space polynomial or larger, it changes the current result and, according to the new definition, our simulation requires logarithmic space.

Finally, we have shown that rewriting input objects, while not exploited in other simulations [2], is essential when less than a logarithmic number of membrane labels is present. In fact, distinguishing two inputs is not possible when the input objects are only moved around in the membrane structure, even when no restriction on the space are present.

In the future we plan to investigate the relationship between the size of the set of objects Γ and the set of membrane labels Λ. It would be interesting to understand if we can easily exchange the way the information is distributed between the two sets according to the new definition of space (Definition 5).

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. Theoretical Computer Science 529, 69–81 (2014)
2. Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes (2014), submitted
3. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
4. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
5. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. International Journal of Computers, Communications & Control 4(3), 301–310 (2009)
6. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. International Journal of Foundations of Computer Science 22(1), 65–73 (2011)
7. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing, 13th International Conference, CMC 2012, Lecture Notes in Computer Science, vol. 7762, pp. 342–357. Springer (2013)

Extending SNP Systems Asynchronous Simulation Modes in P-Lingua

Luis F. Macías-Ramos[1], Tao Song[2], Linqiang Pan[2,⋆], Mario J. Pérez-Jiménez[1]

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

[2] Key Laboratory of Image Information Processing and Intelligent Control
School of Automation
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
taosong@hust.edu.cn, lqpan@mail.hust.edu.cn

Summary. Spiking neural P systems (SN P systems for short) is a developing field within the P systems world. Inspired by the neurophysiological structure of the brain, these systems have been subjected to many extensions in recent years, many of them intended to "somewhat" incorporate more and more features inspired by the functioning of the living neural cells. Although when first introduced in [8] SN P systems were considered to work in synchronous mode, it became clear that considering non-synchronized systems would be rather natural both from both from a mathematical and neuro-biological point of view. Asynchronous variants of these systems were introduced in [4], setting up a scenario where even if a neuron had enabled rules ready to fire, such rules non-deterministically could be not applied. Once new theoretical variants are defined, providing simulation software tools enables experimental study and validation of the proposed models. One more than promising developing branch comprises the use of parallel architectures, concretely GPUs, that provide efficient implementations [1, 2, 7, 3]. One drawback of this approach, due to the inherent constraints of the GPUs programming model, is a relatively long development cycle to extend existing variants. At the expense of sacrificing efficiency for expressivity, other alternatives involving sequential approaches can be considered. Within this trend, P–Lingua [5, 6, 15] offers the high flexibility of the Java programming language as well as a general acceptance within the Membrane computing community. P–Lingua affords a standard language for the definition of P systems. Part of the same software project, *pLinguaCore* library provides particular implementations of parsers and simulators for the models specified in P–Lingua. Support for simulating SN P systems in P–Lingua was introduced in [9]. In that version all (synchronous and asynchronous) "working modes" considered in [14] were implemented. Since then, new asynchronous variants have appeared. In this paper we present a brand

⋆ Corresponding author.

new extension of P–Lingua related to asynchronous SN P systems, in order to incorporate simulation capabilities for limited asynchronous SN P systems, introduced in [12], and asynchronous SN P systems with local introduced respectively in [16].

1 Introduction

SN P systems were introduced in [8] in the framework of Membrane computing [13] as a new class of computing devices which are inspired by the neurophysiological behaviour of neurons sending electrical impulses (spikes) along axons to other neurons.

An SN P system consists of a set of neurons placed as nodes of a directed graph (called the *synapse graph*). Each neuron contains a number of copies of a single object type, the *spike*. Rules are assigned to neurons to control the way information flows between them, i.e. rules assigned to a neuron allow it to send spikes to its neighbouring neurons. SN P systems usually work in a synchronous mode, where a global clock is assumed. In each time unit, for each neuron, only one of the applicable rules is non-deterministically selected to be executed. Execution of rules takes place in parallel amongst all neurons of the system. Nevertheless both from a mathematical and neurological point of view, it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory. If a neuron has an enabled rule in a given time unit, the neuron chooses non-deterministically whether to fire (or not) the rule. Of course, new spikes can come into the neuron when the rule is not fired, rendering it non-applicable. If the rule is still applicable through successive time instants, it can fire at any time, independently of how much time has passed since it first became applicable. Asynchronous SN P systems (ASNPS for short) were introduced in [4] and, as described in [14], the validity of a computation in these systems can be restricted by specifying pre-defined values over the number of spikes for the neurons in the system at halting time. If the computation is considered invalid the output of the system is discarded.

In the systems mentioned above any neuron works asynchronously, in a independent way with respect to the others, such that there is no restriction with respect to the number of successive time instants that the neuron can hold firing of an enabled rule. These two characteristics of the "asynchronous mode" can be overridden. In [12], limited asynchronous SN P systems (LASNPS for short) are introduced. In this variant, a global bound $b \geq 2$ (equal for all neurons) is defined and determines the maximum number of computation steps that a neuron can choose not to fire an enabled rule. On the other hand, in [16] asynchronous SN P systems with local synchronization (ASNPSLS for short) are introduced. In this variant, independence amongst neurons is dropped. Local synchronization enables pre-defining a collection of local synchronizing sets contained in the power set of neurons of the system. When a neuron fires (after deciding not to fire an unbounded number of steps) all the neurons placed in the same local synchronizing set fire immediately.

There exists biological motivation for considering these variants, thus also making them a suitable target to be implemented within a simulation framework like P–Lingua. With respect to LASNPS, in a biological system, if a long enough time interval is given, an enabled chemical reaction will conclude within the given time interval. So it is natural to impose a bound on the time interval in which a spiking rule remains unused. With respect to ASNPSLS, in a biological neural system, motifs with 4-5 neurons and communities with 12-15 neurons, associated with some specific functioning are rather common. Neurons from the same motif or community will work synchronously to cooperate with each other. That is, in a biological system, neurons work globally in an asynchronous way, but synchronously at a local level. Said level is represented by the local synchronizing sets.

In this paper a new extension of SN P systems simulator included in the P–Lingua framework is presented which enables simulation for LASNPS and ASNPSLS. The paper is structured as follows. Section 2 is devoted to introducing background concepts: specifications of SN P systems working in synchronous and asynchronous (normal, limited and local) modes are introduced in an informal way. Section 3 covers the P–Lingua syntax for the new introduced modes. Section 4 presents some P–Lingua files which exemplifies how to define the different models in P–Lingua specification language. Finally, Section 5 shows the corresponding simulation algorithms for the aforementioned models. Section 6 covers conclusions and future work.

2 Preliminaries

In this section we introduce, in an informal way, SN P systems model in their original synchronous form and, subsequently, asynchronous extensions corresponding to limited asynchronous SN P systems, introduced in [12], and asynchronous SN P systems with local synchronization, introduced in [16].

2.1 Spiking neural P systems

SN P systems can be considered a variant of P systems, corresponding to a shift from *cell-like* to *neural-like* architectures. In these systems, cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. Contents of each neuron consist of a number of copies of a single object type, called the *spike*. Every neuron may also contain a number of *firing rules* and *forgetting rules*. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses, *spikes*, which are accumulated at the target neuron, consuming some of their own spikes (in a quantity at least equal to the number of spikes fired). Forgetting rules imply only consuming spikes, while no one is sent to the neighbouring neurons.

The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron

can use one of its rules, then one of such rules must be used. If two or more rules can be applied, then only one of them is non-deterministically chosen. Thus, the rules are used in a sequential way in each neuron, but neurons function in parallel with respect to each other. Let us notice that, as it usually happens in Membrane Computing, a global clock is assumed, marking the time for the whole system, and hence the functioning of the system is synchronized. Also asynchronous scenarios can be considered as shown in [14] and discussed below.

When a cell sends out spikes it becomes "closed" (inactive) for a specified period of time. During this period, the neuron does not accept new inputs and cannot "fire" (that is, cannot emit spikes). The lapse of time required for the rule to be fired is called the "delay" of the rule, which can be any natural number $d \geq 0$. Only firing rules can have a non-negative delay, while forgetting rules have always a delay zero. Let us notice that when $d = 0$, the neuron immediately becomes "open" (active) after firing, being able to send and receive spikes again, thus never being "really closed".

The *configuration* of the system is described by its topological structure (which is constant along computations when not considering division or budding rules) and the number of spikes associated with each neuron. Using the rules as described above, it is possible to define *transitions* among configurations. Any (maximal) sequence of transitions starting in the initial configuration is called a *computation*. A computation *halts* if it reaches a configuration where all neurons are open and no rule can be used.

Further reading about this model, along with a formal specification, can be found in [8].

2.2 Asynchronous SN P systems

In asynchronous SN P systems even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used. The neuron may choose to remain unfired, maybe receiving spikes from the neighbouring neurons. The unused rule may be used later, as long as it stays enabled, without any restriction on the interval during which it has remained unused. If the new spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now).

Further reading about these systems, along with a discussion on their power can be found in [4].

2.3 Limited asynchronous SN P systems

In asynchronous SN P systems an enabled rule not used at a certain instant may be used later, as long as it stays enabled, without any restriction on the interval during which it has remained unused. Nevertheless, from the biological point of view it is convenient to consider a boundary on the number of time units that such rule remains unfired, since in nature given a long enough time interval, an enabled

chemical reaction will conclude within such interval. Following this, in limited asynchronous SN P systems, a single global upper bound $b \geq 2$ (equal for all neurons) is defined on time intervals. If a rule in neuron σ_i is enabled at step t and neuron σ_i receives no spike from step t to step $t+b-2$, then this rule can and must be applied at a step in the next time interval b (that is, at a non-deterministically chosen step from t to $t+b-1$). If the enabled rule in neuron σ_i is not applied, and neuron σ_i receives new spikes, making the rule non-applicable, then computation continues in the new circumstance (maybe other rules are enabled now).

Further reading about these systems, along with a discussion on their universality can be found in [12].

2.4 Asynchronous SN P systems with local synchronization

In asynchronous SN P systems an unused rule may be used later, without any restriction with respect to the functioning (also asynchronous) of the other neurons. Nevertheless, from the biological point of view it is convenient to consider interrelation between neurons in terms of synchronicity. In a biological neural system, small groups of neurons, associated with some specific functioning, are rather common. Neurons within these communities work synchronously to cooperate with each other, while globally working in an asynchronous way with respect to "unrelated" neurons in the system. To model this behaviour in neural-like systems, asynchronous SN P systems with local synchronization are introduced. In these systems, a family of sets (called *ls-sets*) of locally synchronous neurons $Loc = \{loc_1, loc_2, \ldots, loc_l\} \subseteq \mathcal{P}(\{\sigma_1, \sigma_2, \ldots, \sigma_m\})$ is defined ($\mathcal{P}(\{\sigma_1, \sigma_2, \ldots, \sigma_m\})$ being the power set of $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$).

Given neurons in the same ls-set loc_j, if one of these neurons fires, then all neurons in loc_j that have enabled rules should fire. Of course, it is possible that all neurons from loc_j remain unfired even if they have enabled rules. That is, all neurons from loc_j may remain still, or all neurons from loc_j with enabled rules fire at a same step (of course, neurons without enabled rules cannot fire). Hence, neurons work asynchronously at the global level, while working synchronously within each ls-set.

Further reading about these systems, along with a discussion on their universality can be found in [16].

3 P–Lingua Syntax for LASNPS and ASNPSLS

In [6], a Java library called *pLinguaCore* was presented under GPL license. The library provides parsers to handle input files, built–in simulators to generate P System computations and it is able to export several output file formats that represent P systems. *pLinguaCore* is not a closed product because developers with knowledge of Java can add new components to the library, thus extending it. Milestone releases can be downloaded from `http://www.p-lingua.org` while

releases containing latest developments can be found within the distribution of MeCoSim (`http://www.p-lingua.org/mecosim/`). At the time of this writing, the extensions related to the present paper have not been included in a milestone release, so interested readers may refer to the MeCoSim distribution.

This section introduces several recently developed P–Lingua simulators for SN P systems. Support for SN P systems in P–Lingua was introduced in [9], covering the basic model along with neuron division and budding rules as well as asynchronous mode, as originally introduced. Following this, in [10], partial simulation of SN P systems with functional astrocytes (also defined in [10]) was introduced. Finally, simulators for SN P systems with "hybrid" (excitatory and inhibitory) astrocytes and SN P systems with anti-spikes were presented in [11].

In what follows, P–Lingua syntax for defining both LASNPS and ASNPLS is shown. P–Lingua syntax for the other SN P system variants is not covered here, but can be found in the cited papers.

3.1 P–Lingua syntax for limited asynchronous SN P systems

In LASNPS, a global upper bound $b \geq 2$ is defined for all rules. Consequently, a new instruction has been included into P–Lingua to define such upper bound, extending the existing model specification framework for Spiking Neural P systems. Thus, that instruction can be used only when the source P–Lingua files defining the corresponding models begin with the following sentence:

```
@model<spiking_psystems>
```

while also requiring the right asynchronous mode to be set to with the following sentence:

```
@masynch = 3;
```

The instruction to define the global upper bound is:

```
@mboundall = b;
```

where:

- b is the global upper bound, with $b \geq 2$.

3.2 P–Lingua syntax for asynchronous SN P systems with local synchronization

In ASNPLS, a local synchronizing set is defined, consisting of a collection of sets (called ls-sets) that determines which neurons should fire synchronously. Consequently, a new instruction has been included into P–Lingua to define such set, extending the existing model specification framework for Spiking Neural P systems. Thus, that instruction can be used only when the source P–Lingua files defining the corresponding models begin with the following sentence:

```
@model<spiking_psystems>
```

while also requiring the right asynchronous mode to be set to with the following sentence:

```
@masynch = 4;
```

The instruction to define the local synchronizing set is:

```
@mlocset = {ls-1, ls-2, ..., ls-h, ..., ls-m};
```

where:

- ls-h = $\{\sigma_{h,1}, \ldots, \sigma_{h,u_h}\}$ is each one of the ls-sets containing a non-empty collection of membrane labels.

4 Examples

This section is devoted to consider examples dealing with both LASNPS and ASNPSLS. Two sets of examples are presented, one per variant. For each set, a formal specification is presented, followed by a link to the corresponding P–Lingua code. To conclude, an analysis of the functioning of the systems is shown, along with statistical data referred to the output of the systems after several simulations within MeCoSim are performed.

4.1 Asynchronous SN P systems with local synchronization

Consider an asynchronous SN P system with local synchronization Π shown in Figure 1.

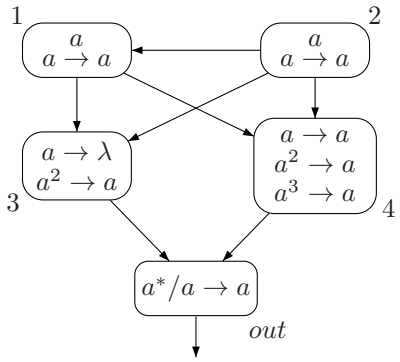


Fig. 1. An example of an asynchronous SN P system with local synchronization Π

The system Π consists of five neurons with labels 1, 2, 3, 4 and *out*. Initially, neurons σ_1 and σ_2 have one spike inside, and other neurons contain no spike. The formal definition of system Π is as follows:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_{out}, Loc, syn), \text{ where}$$

- $\sigma_1 = (1, R_1)$ with $R_1 = \{a \to a\}$;
- $\sigma_2 = (1, R_2)$ with $R_2 = \{a \to a\}$;
- $\sigma_3 = (0, R_3)$ with $R_3 = \{a \to \lambda, a^2 \to a\}$;
- $\sigma_4 = (0, R_3)$ with $R_4 = \{a \to a, a^2 \to a, a^3 \to a\}$;
- $\sigma_{out} = (0, R_{out})$ with $R_{out} = \{a^*/a \to a\}$;
- Loc is the family of sets of locally synchronous neurons;
- $syn = \{(1,3), (1,4), (2,1), (2,3), (2,4), (3, out), (4, out)\}$;
- *out* indicates the output neuron.

To complete the definition of the system, specifying Loc is required. Four cases are considered:

a) $Loc = \emptyset$
b) $Loc = \{\{\sigma_1, \sigma_2\}\}$
c) $Loc = \{\{\sigma_1, \sigma_3\}\}$
d) $Loc = \{\{\sigma_1, \sigma_2\}, \{\sigma_3, \sigma_4\}\}$

The P–Lingua code corresponding to this system can be found at: `http://www.p-lingua.org/examples/SNPSLocalSynch.pli`.

This code is parameterised, allowing execution of each one of the four Π system variants depending on parameter *c*.

Next, we are going to analyse functioning of system Π for each one of the aforementioned cases.

Case a) $Loc = \emptyset$.

In this case, neurons can fire at any time when having enable rules. Output of the system is $\{1, 2, 3, 4\}$.

A table showing the results of the execution of the system after 100 simulations with MeCoSim can be found below.

spikes	ratio
1	39%
2	20%
3	36%
4	5%

Table 1. Ratio of output spikes for case a)

Case b) $Loc = \{\{\sigma_1, \sigma_2\}\}$.

In this case, neurons σ_1, σ_2 fire at the same moment due to local synchronization. After any of the former neurons fires, neuron σ_1 contains 1 spike,

while neurons σ_3, σ_4 contain 2 spikes each. From this point, we have the following cases:

1. Neuron σ_1 fires before neurons σ_3, σ_4. After this, neuron σ_3 contains 3 spikes, so cannot fire, while neuron σ_4 also containing 3 spikes will eventually send out 1 spike. Thus, the output of the system in this case is $\{1\}$.
2. Neuron σ_3 fires before neurons σ_1, σ_4. Spike from neuron σ_3 reaches σ_{out}. From here, any spike passing through σ_3 will be lost, with either 1 or 2 spikes getting to σ_{out} from neuron σ_4. Consequently, the output of the system will be $\{2, 3\}$.
3. Neuron σ_4 fires before neurons σ_1, σ_3. Thus, 1 spike reaches neuron σ_{out} from σ_4. If σ_1 fires before σ_3, this neuron becomes blocked and cannot fire any more, with 1 spike more coming to neuron σ_{out} from σ_4. In any other case, σ_3 sends another spike to σ_{out} and loses the next spike that receives, while σ_4 getting 1 spike that will be sent to σ_{out} eventually. Consequently, the output of the system will be $\{2, 3\}$.
4. Neurons σ_1, σ_3 fire together before σ_4. In this case, σ_3 sends 1 spike to σ_{out}, while the next spike coming from σ_1 will be lost. Finally, σ_4 consumes 3 spikes and sends 1 spike to σ_{out}. Consequently, the output of the system will be $\{2\}$.
5. Neurons σ_1, σ_4 fire together before σ_3. In this case, 1 spike reaches σ_{out} from σ_4, while σ_3, containing 3 spikes, gets blocked. Finally, σ_4 sends 1 spike to σ_{out}. Consequently, the output of the system will be $\{2\}$.
6. Neurons σ_3, σ_4 fire together before σ_1. In this case, 2 spikes reach σ_{out} from σ_3, σ_4 each. Following this σ_1 fires 1 spike that reaches σ_3, σ_4. Spike in σ_3 will be lost while the one in σ_4 will reach σ_{out}. Consequently, the output of the system will be $\{3\}$.
7. Neurons $\sigma_1, \sigma_3, \sigma_4$ fire together. This case is similar to the previous one. Consequently, the output of the system will be $\{3\}$.

As a result of all of this, output of the system in this case is $\{1, 2, 3\}$.

A table showing the results of the execution of the system after 100 simulations with MeCoSim can be found below.

spikes	ratio
1	17%
2	38%
3	25%
4	0%

Table 2. Ratio of output spikes for case b)

Case c) $Loc = \{\{\sigma_1, \sigma_3\}\}$.

In this scenario, we can consider the following cases:

a) Neuron σ_2 fires before σ_1. In this case, after neuron σ_2 fires, σ_1 contains 2 spikes, being unable to continue working, neuron σ_3 contains 1 spike, that will

be lost, and σ_4 contains 1 spike, that will eventually reach σ_{out}. Thus, in this case the output of the system is $\{1\}$.

b) Neurons σ_1, σ_2 fire at the same time. In this case, after the firing, σ_1 contains 1 spike, while σ_3, σ_4 contain 2 spikes each. From here, we have the following cases:
   1. Neurons σ_1, σ_3 fire before σ_4. In this case, neuron σ_{out} gets 1 spike from σ_3, while neuron σ_1 sends 1 spike to σ_3, that will be lost, and σ_4. Having 3 spikes, σ_4 sends 1 spike to σ_{out}. Consequently, the output of the system will be $\{2\}$.
   2. Neurons σ_1, σ_3 fire at the same time as σ_4. This case is similar to the previous one, with σ_{out} getting 2 spikes from σ_4. Consequently, the output of the system will be $\{3\}$.
   3. Neurons σ_1, σ_3 fire after σ_4. This case is similar to the previous one. Consequently, the output of the system will be $\{3\}$.

c) Neuron σ_1 fires before σ_2. In this case, after neuron σ_1 fires, $\sigma_2, \sigma_3, \sigma_4$ contain only 1 spike each. We have the following cases from here:
   1. Neuron σ_2 fires before any other neuron. In this case, neuron σ_1 gets 1 spike, while neurons σ_3, σ_4 contain 2 spikes each. As we showed before, the output of the system will be $\{2, 3\}$.
   2. Neuron σ_3 fires before any other neuron. This spike is lost. From this point, all the spikes passing through σ_3 will be lost, with σ_4 being able to fire up to 3 times. Consequently, the output of the system will be $\{1, 3\}$.
   3. Neuron σ_4 fires before any other neuron. This spike reaches σ_{out}. At this point, σ_2, σ_3 contain 1 spike each. From here:
      - Neuron σ_2 fires first. At this point, σ_1, σ_4 contain 1 spike each, while σ_3 contains 2 spikes. The following cases are possible:
        * Neurons σ_1, σ_3 fire before σ_4. In this case, σ_3 only sends 1 spike to σ_{out} (others are lost), while σ_4 consuming 2 spikes and sending 1 spike to σ_{out}. Consequently, the output of the system will be $\{3\}$.
        * Neurons σ_1, σ_3 fire at the same time as σ_4. This case is similar to the previous one with σ_4 sending an additional spike to σ_{out}. Consequently, the output of the system will be $\{4\}$.
        * Neurons σ_1, σ_3 fire after σ_4. This case is similar to the previous one. Consequently, the output of the system will be $\{4\}$.
      - Neuron σ_3 fires first, resulting in 1 spike being lost. After neuron σ_2 fires, $\sigma_1, \sigma_3, \sigma_4$ contain 1 spike each. The following cases are possible:
        * Neurons σ_1, σ_3 fire before σ_4. In this case, all spikes involving σ_3 are lost, while σ_4 ends consuming 2 spikes and sending 1 spike to σ_{out}. Consequently, the output of the system will be $\{2\}$.
        * Neurons σ_1, σ_3 fire at the same time as σ_4. This case is similar to the previous one with σ_4 sending an additional spike to σ_{out}. Consequently, the output of the system will be $\{3\}$.
        * Neurons σ_1, σ_3 fire after σ_4. This case is similar to the previous one. Consequently, the output of the system will be $\{3\}$.

- Neuron σ_2, σ_3 fire at the same time. This case similar to the previous one. Consequently, the output of the system will be $\{2, 3\}$.

4. Neurons σ_2, σ_3 fire together and before σ_4. Spike in σ_3 is lost, while σ_2 sends 1 spike to $\sigma_1, \sigma_3, \sigma_4$, resulting in σ_1, σ_3 containing 1 spike each while σ_4 contains 2 spikes. The spike in σ_3 will be lost in any case. If σ_1, σ_3 fire before σ_4, 2 spikes are stored in σ_4, thus, only 1 spike is sent to σ_{out}. In other case, 2 spikes reach σ_{out}. Consequently, the output of the system will be $\{1, 2\}$.
5. Neurons σ_2, σ_4 fire together and before σ_3. In this case, 1 spike reaches σ_{out} from σ_4, while the spike sent by σ_2 results in σ_1, σ_4 containing 1 spike each while σ_3 contains 2 spikes. This case is analogous to the previous one taking into account that the first rule applied in σ_3 is the firing rule. Consequently, the output of the system will be $\{3, 4\}$.
6. Neurons σ_3, σ_4 fire together and before σ_2. Spike in σ_3 is lost, while spike in σ_4 reaches σ_{out}. After σ_2 fires, $\sigma_1, \sigma_3, \sigma_4$ contain 1 spike each. Spikes passing through σ_3 will be lost and either another 1 or 2 spikes can get to σ_{out} from σ_4. Consequently, the output of the system will be $\{2, 3\}$.
7. Neurons $\sigma_2, \sigma_3, \sigma_4$ fire together. This case is analogous to the previous one. Consequently, the output of the system will be $\{2, 3\}$.

As a result of all of this, output of the system in this case is $\{1, 2, 3, 4\}$.

A table showing the results of the execution of the system after 100 simulations with MeCoSim can be found below.

spikes	ratio
1	35%
2	29%
3	33%
4	3%

Table 3. Ratio of output spikes for case c)

Case d) $Loc = \{\{\sigma_1, \sigma_2\}, \{\sigma_3, \sigma_4\}\}$.

Neurons σ_1, σ_2 will fire at the same moment (due to the local-synchronization) sending 2 spikes to neurons σ_3, σ_4 respectively. At that moment, neuron σ_1 receives 1 spike from neuron σ_2. With 2 spikes inside, neurons σ_3, σ_4 will fire at the same moment. If they fire before neuron σ_1, then σ_{out} receives 2 spikes first, and then 1 spike more from σ_4 (the spike in σ_3 is lost), thus receiving in total 3 spikes. The case in which all neurons fire at the same time is similar to the previous one. To conclude, if neuron σ_1 fires before neurons σ_3, σ_4, then neuron σ_3 cannot fire, and σ_4 will consume 3 spikes and send 1 spike to σ_{out}.

Consequently, output of the system is $\{1, 3\}$.

A table showing the results of the execution of the system after 100 simulations with MeCoSim can be found below.

spikes	ratio
1	15%
2	0%
3	85%
4	0%

Table 4. Ratio of output spikes for case d)

Conclusion
The computation result of Π is $\{1, 2, 3, 4\}$.

4.2 Limited asynchronous SN P systems

Consider a limited asynchronous SN P system Π' shown in Figure 2.

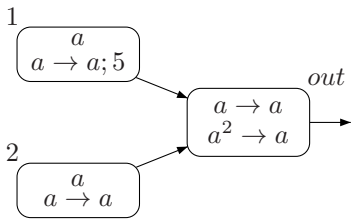


Fig. 2. An example of a limited asynchronous SN P system Π'

The system Π consists of three neurons with labels 1, 2 and *out*. Initially, neurons σ_1 and σ_2 have one spike inside with σ_{out} containing no spike. The formal definition of system Π' is as follows:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_{out}, b, syn), \text{ where}$$

- $\sigma_1 = (1, R_1)$ with $R_1 = \{a \to a; 5\}$;
- $\sigma_2 = (1, R_2)$ with $R_2 = \{a \to a\}$;
- $\sigma_{out} = (0, R_{out})$ with $R_{out} = \{a \to a, a^2 \to a\}$;
- b is a single upper bound on time intervals, valid for all rules;
- $syn = \{(1, out), (2, out)\}$;
- *out* indicates the output neuron.

To complete the definition of the system, specifying upper bound $b \geq 2$ is required. Two cases are considered:

a) $b = 2$
a) $b = 4$

The P–Lingua code corresponding to this system can be found at: `http://www.p-lingua.org/examples/SNPSLimited.pli`.

This code is parameterised, allowing execution of each one of the two Π' system variants depending on parameter b.

Next, we are going to analyse functioning of system Π' for each one of the aforementioned cases.

Case a) $b = 2$.

Initially, neurons σ_1, σ_2 have 1 spike inside. Neuron σ_2 will fire no later than step b = 2, sending 1 spike to neuron σ_{out}. With 1 spike inside, σ_{out} will fire before step 2b = 4, sending 1 spike to the environment. Neuron σ_1 can fire at step 1 or 2 (due to the fact that the interval bound b is 2), and sends 1 spike to neuron σ_{out} at step 6 or 7. In this way, neuron σ_{out} will emit 2 spikes into the environment.

Consequently, output of the system is $\{2\}$.

A table showing the results of the execution of the system after 100 simulations with MeCoSim can be found below.

spikes	ratio
1	0%
2	100%
3	0%
4	0%

Table 5. Ratio of output spikes for case a)

Case b) $b = 4$.

Initially, neurons σ_1, σ_2 have 1 spike and will fire at any step no later than step b = 4. Following this, if neuron σ_1 fires at step 1, it sends 1 spike to neuron σ_{out} at step 6 and so on. On the other hand, if neuron σ_2 fires at step 4, it sends 1 to neuron σ_{out} at such step. In this case, after neuron σ_{out} receives the first spike at step 4, it will fire at any step no later than step 8 (due to the fact that the upper bound b is 4). As a result of all of this, we have to cases:

a) If σ_{out} stays inactive later than step 6, it will accumulate 2 spikes inside, and send 1 spike into the environment.
b) If σ_{out} fires before step 6, it will send 2 spikes to the environment.

In this condition, The system can generate numbers 1 and 2. In a similar way other computations of the system can be checked.

Consequently, output of the system is $\{1, 2\}$.

A table showing the results of the execution of the system after 100 simulations with MeCoSim can be found below.

Therefore, the computation result of Π' is $\{1, 2\}$.

spikes	ratio
1	2%
2	98%
3	0%
4	0%

Table 6. Ratio of output spikes for case b)

5 Simulation Algorithm

In what follows a P–Lingua based simulation algorithm for LASNPS and ASNPSLS is shown in pseudo-code form. This algorithm is a revision from the one included in the foundational paper on simulating SN P systems in P–Lingua [9]. Consequently, it generates one possible computation for a SN P system with an initial configuration C_0 containing n neurons $m_1, \ldots, m_n$. Let us recall that when working with recognizer P systems all computations yield the same answer (confluence).

The simulation algorithm is structured in six stages:

I. Initialization
In this stage the data structures needed to perform the simulation are initialized.
II. Selection of rules
In this stage the set of rules to be executed in the current step is calculated.
III. Build execution sets
In this stage the rules to be executed are split into different sets, according to their kind.
IV. Execute division and budding rules
In this stage division and budding rules are executed. The execution is performed in two phases: In the first one, new neurons are calculated out of existing neurons by applying budding and division rules. In the second one additional synapses are introduced according to the synapse dictionary.
V. Execute spiking rules
In this stage execution of spiking rules is performed.
VI. Ending
In this stage the current configuration is updated with the newly calculated one and the halting condition is checked (no more rules are applicable).

The simulation algorithm follows.

I. Initialization

1. Let C_t be the current configuration
2. Let $M_{sel} \equiv \emptyset$ be a set of membranes who are susceptible of executing a rule in the current computation step
3. Let m_0 be a virtual membrane (with label 0) representing the environment

II. Selection of rules

1. Each membrane m_i stores the following elements:
   - last rule r_i selected to be executed in a previous step for that membrane (initially the void rule)
   - an integer decreasing-only counter d_i, that stores the number of steps left for the membrane to open and fire in case r_i is a firing rule (initially zero).
   - an integer decreasing-only counter b_i, that stores the number of steps left for the membrane to decide if its current selected rule can be freely chosen to fire or not (initially zero; if zero then it should fire in this step).

   For each membrane m_i, do
   a) If m_i is closed as a result of being involved in the execution of a budding or division rule, then open m_i (let $d_i = 0$) and clear its rule r_i
   b) If the simulator is working on limited asynchronous mode and each one of the following statements is true
      i. $b_i > 0$
      ii. r_i is not void
      iii. r_i is active
      iv. r_i can be applied over m_i

      Then
      i. Decrease the counter b_i
      ii. Add m_i to M_{sel}
      iii. Go to process the next membrane
   c) If m_i is closed as a result of being involved in the execution of a firing rule (thus r_i is a firing rule) then
      i. Decrease the counter d_i
      ii. Add m_i to M_{sel}
      iii. Go to process the next membrane
   d) Let $S_i \equiv \emptyset$ be the set of possible rules to be executed over m_i
   e) For each rule r_j with label j do
      i. If r_j is active and can be executed over m_i then add r_j to S_i
   f) If S_i is empty then
      i. Set b_i to zero
      ii. Go to process the next membrane
   g) Select non deterministically a rule r_k from S_i
   h) Set r_k as the new selected rule for m_i
   i) If r_k is a firing rule, update the counter d_i accordingly
   j) Restart the counter b_i to the global upper bound b
   k) Add m_i to M_{sel}
   l) Clear S_i
2. If M_{sel} is not empty and the simulator operates in Sequential Mode then
   a) Select a membrane m_s from M_{sel} according to the Sequential Mode
   b) Clear M_{sel}
   c) Add m_s to M_{sel}

III. Build execution sets

1. Let $Division \equiv \emptyset$ be the set that stores the membranes having a division rule selected to be executed in the current step
2. Let $Budding \equiv \emptyset$ be the set that stores the membranes having a budding rule selected to be executed in the current step
3. Let $Spiking \equiv \emptyset$ be the set that stores the membranes having a spiking rule selected to be executed in the current step (or susceptible to be executed in the case of firing rules with delays)
4. Let $Available \equiv \emptyset$ be the set that stores the membranes having a rule selected to be executed in the current step
5. Let $toFire \equiv \emptyset$ be the set that stores the membranes having a rule that will be fired in the current step
6. Let $toFireAux \equiv \emptyset$ be an auxiliary set
7. Let $locProc \equiv \emptyset$ be the set that stores the membranes that have been processed in terms of local synchronizing (that is, each one of their neighbours have been marked to fire immediately and have been processed also)
8. For each membrane m_i from M_{sel} do
   a) Let r_i be the selected rule for m_i
   b) If r_i is a division rule then add m_i to $Division$
   c) If r_i is a budding rule then add m_i to $Budding$
   d) If r_i is a spiking rule then add m_i to $Spiking$
   e) Add m_i to $toFireAux$ if and only if the call to procedure $DecideToFire(m_i, r_i)$ yields true.
   f) If the simulator is operating in asynchronous mode with local synchronization then add m_i to $Available$
9. Add all membranes from $toFireAux$ into $toFire$
10. If the simulator is operating in asynchronous mode with local synchronization then for each membrane m_i from $toFireAux$ do
    a) Call the recursive procedure $Process(m_i, Available, locProc, toFire)$

Procedure $DecideToFire(m_i, r_i)$

1. Let m_i, r_i be input/output arguments declared consistently as specified above
2. If r_i is a budding rule or a division rule, then return the result of the call to procedure $DecideAsynch(m_i, r_i)$
3. If r_i is a firing rule or a forgetting rule, then
   a) Let d be the delay associated to rule r_i
   b) Let s be the number of steps left for membrane m_i to become open
   c) If $d = 0$ then return the result of the call to procedure $DecideAsynch(m_i, r_i)$
   d) Else
      i. If $d = s$ return the result of the call to procedure $DecideAsynch(m_i, r_i)$
      ii. Else return true

Procedure $DecideAsynch(m_i)$

1. Let m_i be input/output argument declared consistently as specified above
2. Return true if and only if one of the following statements is true

a) the simulator is operating in synchronous mode
b) the simulator is operating in "Standard" asynchronous mode and the truth value "true" is obtained with a probability equal to 0.5
c) the simulator is operating in limited asynchronous mode and either
   c.1) the counter b_i associated to m_i is equal to zero
   c.2) otherwise the truth value "true" is obtained with a probability equal to 0.5 and, subsequently, the counter b_i associated to m_i is set to zero
d) the simulator is operating in asynchronous mode with local synchronization and the truth value "true" is obtained with a probability equal to 0.5

Recursive procedure $Process(m_i, Available, locProc, toFire)$

1. Let $m_i, Available, locProc, toFire$ be input/output arguments declared consistently as specified above
2. If $m_i \in locProc$ then exit
3. Else
   a) Add m_i into $locProc$
   b) If $m_i \in Available$ then
      i. Add m_i into $toFire$
      ii. Let $Affected$ be the set containing all the membranes that should immediately fire in case m_i fires
      iii. For each membrane $m_j \in Affected$, do
         A. Call the recursive procedure $Process(m_j, Available, locProc, toFire)$

IV. Execute division and budding rules

1. Let $Div \equiv \emptyset$ be the set that stores the membranes that are generated as a result of applying a division rule in the current step
2. Let $Bud \equiv \emptyset$ be the set that stores the membranes that are generated as a result of applying a budding rule in the current step
3. For each membrane m_i from $Division$ do
   a) If $m_i \notin toFire$ then go to process the next membrane
   b) Let r_i be the selected rule for m_i: $[E]_i \rightarrow []_j || []_k$
   c) Relabel m_i with the j label, thus from now on we refer to m_j
   d) Create a new membrane m_k and close it
   e) For each incoming edge from some membrane m_p to m_j create a new edge from m_p to m_k
   f) For each outgoing edge from m_j to some membrane m_p create a new edge from m_k to m_p
   g) Add m_j and m_k to Div
4. For each membrane m_i from $Budding$ do
   a) If $m_i \notin toFire$ then go to process the next membrane
   b) Let r_i the selected rule for m_i: $[E]_i \rightarrow []_i / []_j$
   c) Create a new membrane m_j and close it
   d) For each outgoing edge from m_i to some membrane m_p do
      i. Create a new edge from m_j to m_p

ii. Remove the edge from m_i to m_p
e) Create a new edge from m_i to m_j
f) Add m_j to Bud

5. For each membrane m_i from Div create new edges involving m_i according to the synapse dictionary if necessary
6. For each membrane m_i from Bud create new edges involving m_i according to the synapse dictionary if necessary

V. Execute spiking rules

1. For each membrane m_i from $Spiking$ do
   a) If $m_i \notin toFire$ then go to process the next membrane
   b) If m_i is closed then go to process the next membrane
   c) Let r_i be the selected rule for m_i
   d) If r_i is a firing rule of the form $[E/a^c \rightarrow a^p; d]_i$ then
      i. Remove c spikes from the multiset of m_i
      ii. For each membrane m_j connected to m_i by an edge going from m_i to m_j, add p spikes to the multiset of m_j if and only if m_j is open
   e) If r_i is a forgetting rule of the form $[E/a^c \rightarrow \lambda]_i$ then remove c spikes from the multiset of m_i

VI. Ending

1. Let $C_{t+1} = C_t$
2. If M_{sel} is not empty then goto **I**

6 Conclusions and Future Work

In this paper we have shown very recent developments in the field of simulators for SN P systems, concretely P–Lingua based ones. New variants are presented, integrating limited and locally synchronized asynchronous modes. In this sense, a new release of P–Lingua, that extends the previous SN P systems simulator has been developed, incorporating the ability to work with the new implemented models. This new simulator has been included into the library *pLinguaCore* and tested by simulating selected examples provided by experts and referred in Section 4.

At the moment, an extension to incorporate fuzzy reasoning SN P systems is in development. Once this work is done, a desirable feature would be to provide a mechanism for defining arbitrary computable functions, thus fully simulating SNPSFA. Additional elements such as weights might also be incorporated. Also connecting *pLinguaCore* with existing CUDA-based simulators in being considered at present.

Acknowledgements

The authors acknowledge the support of the project TIN2012–37434 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds. T. Song an L. Pan were supported by National Natural Science Foundation of China (61033003, 91130034, and 61320106005).

References

1. Cabarle, F.G., Adorna, H.N., Martínez-Del-Amor, M.A.: Simulating spiking neural P Systems without delays using GPUs. IJNCR 2(2), 19–31 (2011)
2. Cabarle, F.G., Adorna, H.N., Martínez-Del-Amor, M.A., Pérez-Jiménez, M.J.: Spiking neural P System simulations on a high performance GPU platform. Lecture Notes in Computer Science 7017, 99–108 (10/2011 2011), http://www.springerlink.com/content/f490qnv027884g27/, algorithms and Architectures for Parallel Processing ICA3PP Workshops, (ADCN 2011)
3. Cabarle, F.G., Adorna, H.N., Martínez-Del-Amor, M.A., Pérez-Jiménez, M.J.: Improving GPU simulations of spiking neural P Systems. Romanian Journal of Information Science and Technology 15, 5–20 (06/2012 2012), http://www.imt.ro/romjist/Volum15/Number15_1/cuprins15_1.htm
4. Cavaliere, M., Egecioglu, m., Ibarra, O.H., Ionescu, M., Paun, G., Woodworth, S.: Asynchronous spiking neural p systems: Decidability and undecidability. In: Garzon, M.H., Yan, H. (eds.) DNA. Lecture Notes in Computer Science, vol. 4848, pp. 246–255. Springer (2007), http://dblp.uni-trier.de/db/conf/dna/dna2007.html#CavaliereEIIPW07
5. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5391, pp. 187–203. Springer (2008)
6. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An overview of P-Lingua 2.0. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5957, pp. 264–288. Springer (2009)
7. Gheorghe, M., Paun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.): Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers, Lecture Notes in Computer Science, vol. 7184. Springer (2012)
8. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P Systems. Fundam. Inform. 71(2-3), 279–308 (2006)
9. Macías-Ramos, L.F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua based simulator for spiking neural P Systems. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) Int. Conf. on Membrane Computing. Lecture Notes in Computer Science, vol. 7184, pp. 257–281. Springer (2011)
10. Macías-Ramos, L.F., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Functional Astrocytes, 12th international conference on Membrane Computing, accepted paper

11. Macías-Ramos, L.F., Pérez-Jiménez, M.J.: On recent developments in p-lingua based simulators for spiking neural p systems. Asian Conference on Membrane Computing pp. 14–29 (10/2012 2012)
12. Pan, L., Wang, J., Hoogeboom, H.J.: Limited asynchronous spiking neural p systems. Fundam. Inf. 110(1-4), 271–293 (Jan 2011), http://dl.acm.org/citation.cfm?id=2362097.2362116
13. Păun, G.: Computing with membranes (P Systems): An introduction. In: Current Trends in Theoretical Computer Science, pp. 845–866 (2001)
14. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
15. Research Group on Natural Computing, University of Seville: The P–Lingua website. http://www.p-lingua.org
16. Song, T., Pan, L., Păun, G.: Asynchronous spiking neural p systems with local synchronization. Inf. Sci. 219, 197–207 (Jan 2013), http://dx.doi.org/10.1016/j.ins.2012.07.023

Revisiting Sevilla Carpets: A New Tool for the P-Lingua Era

David Orellana-Martín, Carmen Graciani, Miguel Ángel Martínez-del-Amor, Agustín Riscos-Núñez, Luis Valencia-Cabrera

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
dorelmar@gmail.com, {cgdiaz, mdelamor, ariscosn, lvalencia}@us.es

Summary. Sevilla Carpets have already been used to compare different solutions of the Subset Sum problem: either designed in the framework of P systems with active membranes (both in the case of membrane division and membrane creation), and also another one in the framework of tissue-like P systems with cell division.

Recently, the degree of parallelism and other descriptive complexity details have been found to be relevant when designing parallel simulators running on GPUs.

We present here a new way to use the information provided by Sevilla carpets, and a script that allows to generate them automatically from P-Lingua files.

1 Introduction

P systems are massively parallel computing devices, in the sense that their evolution eventually involves a great number of symbol-objects, membranes and rules. Furthermore, if we work with models where the number of membranes can increase along the computation, via creation or division of membranes, then it becomes specially difficult to describe the complexity of the computational process. Such models have actually been investigated largely in the literature, as their ability to generate an exponential number of membranes in polynomial time (making use of their intrinsic parallelism) makes them powerful tools for solving **NP**-complete problems. Indeed, several efficient solutions to these type of problems have been presented (see, e.g. [6, 16, 17, 18] or [19]).

The complexity in *time* (number of cellular steps) of the solutions obtained in this way is polynomial, but it is clear that time is not the unique variable that we need to consider in order to evaluate the complexity of such processes. This fact has been observed previously in the literature of P systems. The first paper related to this issue was [2], where G. Ciobanu, Gh. Păun and Gh. Ştefănescu presented a new way to describe the complexity of a computation in a P system,

the so-called *Sevilla Carpet*, which is an extension of the notion of Szilard language from grammars to the case when several rules are used at the same time.

In [8], 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.

Sevilla Carpets have been adapted to tissue-like models, in order to describe the complexity of their computations (see [4]). There exists also an extension of the definition of Sevilla carpets to a four-dimensional manifold (see [10]) which can be used for a more verbose description of the complexity of a computation of a P system. The graphical representation of this four-dimensional manifold is carried out via projections on three-dimensional spaces.

Comparing two cellular designs that solve the same problem is not an easy task, as there are many ingredients to be taken into account. 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. First, we recall the definition of the Sevilla carpets, together with a list of parameters associated with them. Then, we describe the tool that allows to generate Sevilla carpets automatically from P-Lingua files. Several examples are displayed, and we conclude the paper by providing an overview on the future directions of this ongoing work.

2 Sevilla Carpets

As pointed out in the introduction, the evolution of a P system is usually a too complex process to be evaluated only by the classical parameters from computational complexity measure, *time* and *space*. For instance, we are often interested in other types of descriptive complexity information: size of the alphabet, number of membranes (initially in the system or obtained during the computation), number of rules, etc. Another interesting parameter, especially when running software simulations, is the number of elementary operations (applications of rules) that are performed during the computation.

A possible way to describe the complexity of the evolution of a P system is by means of Sevilla carpets. They were presented in [2] 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., [11, 14] or [20]). The original framework for Szilard

language is the Chomsky hierarchy of grammars, where only one rule is used in each derivation step and, therefore, a derivation can be represented in a natural way as the string of labels corresponding to the used rules (the rule labelling is supposed to be one-to-one, we refer again to [2] for details).

Sevilla carpets are a Szilard-way to describe a computation in a P system, capturing via a bi-dimensional writing the fact that in each evolution step a P system can use not just a single rule, but a multiset of them. More precisely, 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.

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.

In what follows, we shall focus on the third variant (studied in [8]), that is, in each cell of the table we specify the number of applications of the corresponding rule in the considered step. Note that there is a huge amount of data contained in a Sevilla carpet, describing a computation of a P system, but these data are presented in a rough way, just a listing of which rules were applied at each step and how many times.

In order to facilitate reading the whole table just in one glance, we can obtain a three-dimensional representation of it in a natural way, expressing the numbers in each cell over a third axis (see Figure 1).

However, such a three-dimensional picture may not provide significant information by itself, specially in the case of comparing several carpets. In order to be able to "evaluate" the massive amount of information contained in the table of the Sevilla carpet, we need to extract some figures or statistics. The first natural parameters related with Sevilla carpets were defined in [2]: the sum of all the cells in the table (*weight*) and the total amount of cells in the table (*surface*). It is clear that the values of the weight and the surface of a Sevilla carpet give a general intuition on the complexity of the underlying computation. On one hand, the weight measures up the total number of applications of rules along the computation, which corresponds to the intuitive notion of "cost". On the other hand, the surface tells us about the $space \times time$ complexity of the system that is carrying out the computation, as the number of rows is the number of rules of the system

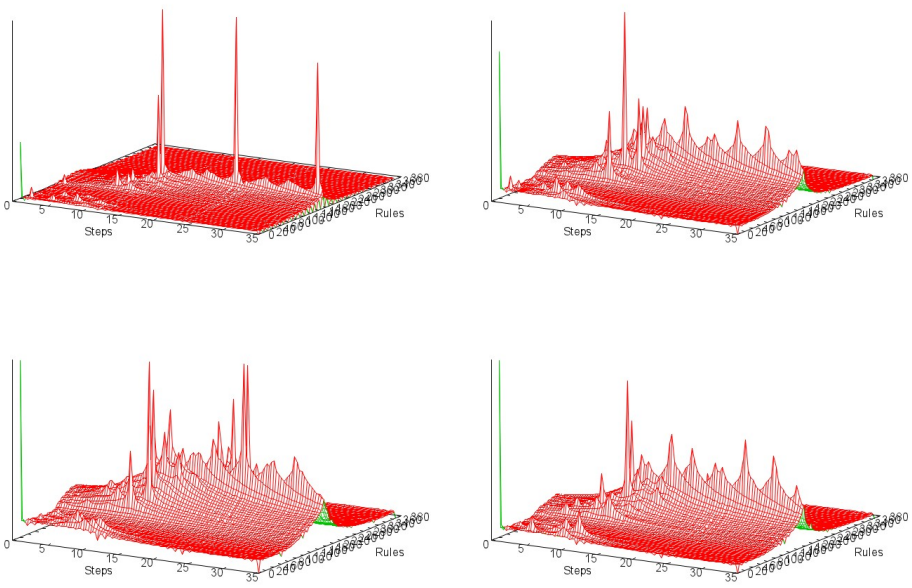


Fig. 1. Sevilla Carpets associated with a solution to SAT using membrane division (running on four different instances)

and the number of columns is the number of cellular steps that the computation performs.

2.1 Parameters for the Descriptive Complexity

The following parameters have been proposed in the literature:

- **Weight:** It is defined in [2] as the sum of all the elements in the carpet, i.e., as the total number of applications of rules along the computation. The weight measures up the total number of applications of rules along the computation, which corresponds to the intuitive notion of "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, was also introduced in [2]. It can be considered as the *potential size* (*space* $\times$ *time* bounds) 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:** Introduced in [8], the height of a Sevilla carpet captures the intuition of a peak in the computation, and it is defined as the maximum number of simultaneous (in one step) applications of any rule all over 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. This parameter was also introduced in [8].
- **Variance:** It is calculated as the sum of the squared differences between the elements of the carpet and the average weight, divided by the surface.
  This parameter was introduced in [10], and it indicates if the points are near or far from the average. That is, a high variance value indicates that there is a very large number of applications of rules performed in a few steps in the computation, while in the rest of the steps the activity can be considered low (see the peaks and valleys in Figure 1). On the other hand, a low variance value leads to think that the *work load* is more balanced.

One of the motivations of this paper is to facilitate the use of the information provided by the Sevilla carpets in the context of GPU-based simulators for P systems (see e.g. [1, 12, 13]). It has been observed that the speed-up obtained by such parallel simulations (with respect to standard sequential simulators) highly depends on how distributed the rule applications are during the simulated computation. Informally speaking, the underlying intuition agrees with the observation from [9]:

> *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 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.*

More precisely, the notion of *GP-systems* (GPU-oriented P systems) is introduced in [12], and also some specific parameters related to the performance of GPU simulations:

- **Density of objects per membrane:** general purpose parallel simulators usually save threads for the whole alphabet, so the more different objects are in the membrane, the higher thread usage.
- **Rule intensity:** some designs include rules associated with auxiliary objects which are only applied once in the whole computation (e.g. counters or synchronization routines). This cannot be parallelized.
- **Communication among membranes:** the skin is executed on the CPU, and every time we need to communicate objects through PCI express bus, this process slows down the process.

2.2 Projections of Sevilla Carpets

The graphical representation in 3D of the Sevilla carpet of a computation provides an intuitive representation of the computational effort associated with each rule in each step. Nevertheless, sometimes it is better to have a more concise representation of this effort. In these cases we can consider the *projections* of the Sevilla carpet. These *projections* are obtained in two different ways:

- By considering the whole number of applications of rules for each step. If the application of a rule has an associated *cost*, this projection will give information about the whole cost of each step.
- By considering for a given rule the whole number of steps in which it has been applied. This provides information about the *utility* of a rule: if we have designed a solution of a problem where several rules are used a low number of times along the computation, we can consider to replace these rules by another rule with the same function.

3 Tool description

In this paper we present a tool that automatically generates Sevilla carpets. The script receives the description of a P system in P-Lingua syntax (a plain text file with .pli extension, we refer to [5, 21] for details), and produces a jpg image of a 3D representation of the Sevilla Carpet associated to (one computation of) the given P system.

In this first version, we have used *python* as programming language, and *gnuplot* for producing the graphical output.

The tool works as follows: first, pLinguaCore library computes a single computation, then the python script parses the results and generates a matrix with the numerical data corresponding to the points of the Carpet. Finally, gnuplot is called and it renders the matrix to a 3D graph that represents the Sevilla carpet associated with this computation.

Note that this represents a notable improvement from previous works, where the process was done manually, after processing the output of simulators for P systems with active membranes written in Prolog ([3, 7]). Now it became a much faster and simpler process, we avoid noise in data due to mistakes when manually building the matrix, and we can cover all the models of P systems included in the P-Lingua framework.

3.1 The algorithm

The parser is designed to work on a text file generated by pLinguaCore library containing verbose information about a computation.

More precisely, the file describes the sequence of configurations, and a list of which rules were applied at each step, and how many times. The format used in such file is as follows:

STEP k:
Rules selected for MEMBRANE ID: x, Label: y, Charge: z
n * #r q
...

where:

- k represents the current step.
- r is the label (usually a number) of the applied rule.
- q is the rule itself.
- n represents the number of times that rule r is applied in step k.

We have thus all the necessary information to generate a matrix M with the data to be plotted.

The parser reads the whole file, paying attention to the lines that have a "Step" statement, or an applied rule, and ignoring the rest.

Whenever a *Step* is found, we move to the next row of the matrix, and whenever a *rule* line is read, we apply the next:

$$M_{current\ step-1,r-1} = M_{current\ step-1,r-1} + n$$

If the rule r, isn't applied in the step k, then we can do:

$$M_{k-1,r-1} = 0$$

From this matrix, we can successfully obtain the x axis (*steps*, the rows of the matrix), the y axis (*rules*, columns of the matrix) and the z axis (*number of times*, values of the matrix).

We can then, with *gnuplot* generate a graph with the data obtained, and create the Sevilla Carpet.

```
STEP: 3

Rules selected for MEMBRANE ID: 1, Label: 2, Charge: 0
1 * #109 d{1}[]'2 --> [d{2}]
1 * #112 [s{1,1} --> s{1,2}]'2

Rules selected for MEMBRANE ID: 2, Label: 2, Charge: 0
1 * #109 d{1}[]'2 --> [d{2}]
1 * #113 [s{2,1} --> s{2,2}]'2
```

Fig. 2. A few lines extracted from a computation file

In the example shown in Figure 2, for step 3 we get the following values in the table: $(3, 109, 2)$, $(3, 112, 1)$, $(3, 113, 1)$, and $(3, r, 0)$ for any other rule label r.

4 Examples

We have done some examples, using .pli files corresponding to solutions to SAT, KNAPSACK, PARTITION and SUBSETSUM designed by means of families of P systems with active membranes.

Here are the results of some of the computations:

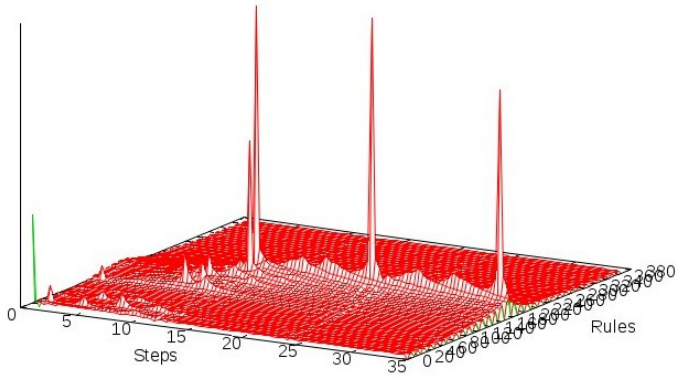


Fig. 3. SAT simulation

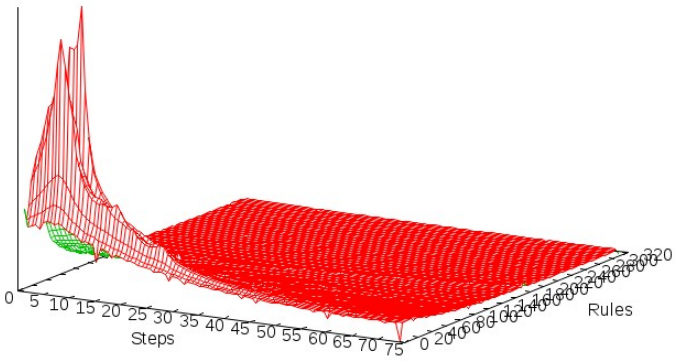


Fig. 4. KNAPSACK simulation

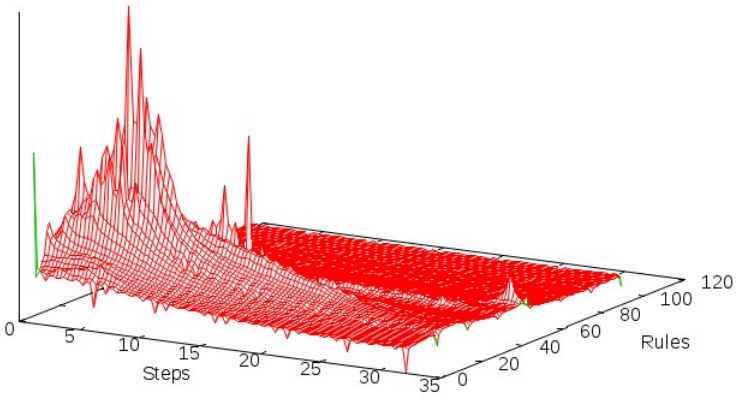


Fig. 5. PARTITION simulation

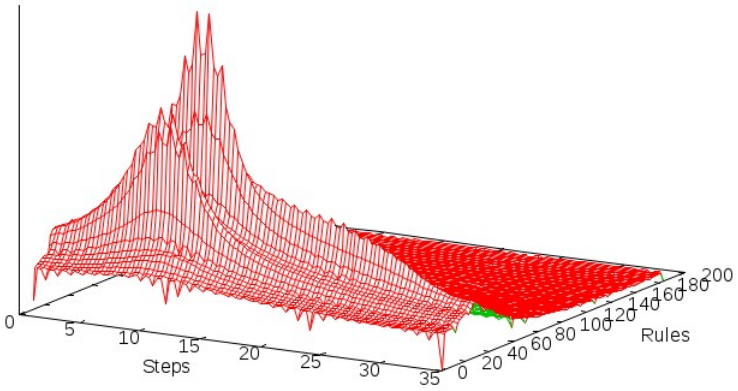


Fig. 6. SUBSETSUM simulation

5 Final Remarks and Future Work

We would like to remark that the information extracted from a Sevilla carpet (and from its associated parameters) should always be interpreted being aware that it only refers to one computation. Nevertheless, we still believe that it is a useful instrument that can be used for example as a guide to find possible refinements on the definition of the simulated P system (e.g. in order to "translate" it into an equivalent GP system), or as an assistant for designing ad-hoc simulators for particular families of P systems (e.g. solutions to hard problems using active membranes)[1].

The next improvement to be added in the near future is to get the values of all parameters together with the graphical 3D representation. Some other customization possibilities can also be considered for the script, like adding options for which kind of Carpet is wanted.

Another interesting possibility is to plug a Sevilla carpet module (adapting the script presented here) into the pLinguaCore library, in such a way that the matrix can be generated on-the-fly as the computation is being simulated, thus avoiding parsing the output file. We are also considering to bring the idea of the script into MeCoSim as an extension. MeCoSim [15, 22] is a general purpose software tool to model, design, simulate, analyze and verify different types of models based on P systems (specially intended for PDP systems). It is a highly customizable tool, allowing to easily configure ad-hoc GUIs for each case study to be modeled. Therefore, it seems reasonable to offer a Sevilla carpet as one possible output that the user can be interested on, together with some other descriptive complexity details (for example, number of membranes generated during the computation).

In the case of probabilistic P systems, it might be interesting to extract several samples of computations and then use the average values in order to generate the Sevilla carpet and the associated parameters. Actually, MeCoSim already includes the possibility to run several computations when working with models for ecosystems designed using PDP systems, and then uses the statistical information gathered to plot the output graphics.

Acknowledgements

The authors acknowledge the support of the Project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain, cofinanced by FEDER funds.

[1] This work direction seems to be worth studying, in the context of the PMCGPU project [23].

References

1. Cecilia, J.M.; Garca, J.M.; Guerrero, G.D.; Martnez-del-Amor, M.A.; Prez-Jimnez, M.J.; Ujaldn, M. The GPU on the simulation of cellular computing models *Soft Computing*, **16** (2), 2012, 231–246.
2. Ciobanu, G.; Păun, Gh.; Ştefănescu, Gh. 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.
3. Cordón-Franco, C.; Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Sancho-Caparrini, F. A Prolog simulator for deterministic P systems with active membranes, *New Generation Computing*, **22** (4), 2004, 349–363.
4. Daz Pernil, D., Gallego-Ortiz, P., Gutirrez Naranjo M.A., Prez Jimnez M.J., Riscos Nez A. Descriptional Complexity of Tissue-like P Systems with Cell Division. In C.S. Calude et al. (eds.) Unconventional Computation. Lecture Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg, 5715 (2009), 168-178.
5. Daz Pernil, D.; Prez–Hurtado, I.; Prez-Jimnez, M.J.; Riscos-Nez, A. A P-lingua programming environment for Membrane Computing. *Lecture Notes in Computer Science*, **5391** (2009), 187–203.
6. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Riscos-Núñez, A. A Fast P System for Finding a Balanced 2-Partition, *Soft Computing*. Springer. To appear.
7. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez, A Simulator for Confluent P Systems. In M. A. Gutiérrez-Naranjo, A. Riscos-Núñez, F. J. Romero-Campero, D. Sburlan (eds.), *Third Brainstorming Week on Membrane Computing* Fénix Editora, Sevilla, 2005, 169–184.
8. 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-Verlag, 2005, 320–330.
9. 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.
10. M. A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Multi-dimensional descriptional complexity of P systems. Journal of Automata, Languages and Combinatorics 12 (2007) 1/2, 167179. A preliminar version in Proceedings of the 7th International Workshop on Descriptional Complexity of Formal Systems, Como, Italy, June 30 - July 2, 2005, pp. 134-145.
11. Mäkinen, E. A Bibliography on Szilard Languages, Dept. of Computer and Information Sciences, University of Tampere, `http://www.cs.uta.fi/reports/pdf/Szilard.pdf`
12. Martnez-del-Amor, M. A. *Accelerating Membrane Systems Simulators using High Performance Computing with GPU* (PhD Thesis), 2013.
13. Martnez-del-Amor, M. A.; Prez-Hurtado, I.; Prez-Jimnez, M.J.; Cecilia J.M.; Guerrero, G.D.; Garca, J.M. Simulating active membrane systems using GPUs. In Gh. Păun et al (eds.) *10th Workshop on Membrane Computing*, 369–384.
14. Mateescu, A. and Salomaa, A. Aspects of Classical Language Theory, in G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages* (vol. 1), Springer-Verlag, Berlin Heidelberg, 1997.

15. Prez-Hurtado, I.; Valencia, L.; Prez-Jimnez, M.J.; Colomer, M.A.; Riscos-Nez, A. MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems. In K. Li, Z. Tang, R. Li, A.K. Nagar, R. Thamburaj (eds.) *Proceedings 2010 IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010)*, IEEE Press, Volume 1, September 23-26, 2010, Changsha, China, ISBN 978-1-4244-6439-5, pp. 637–643.
16. Pérez-Jiménez, M.J.; Riscos-Núñez, A. Solving the Subset Sum Problem by Active Membranes, *New Generation Computing*, Vol. 23, num. 4 (2005), 367-384.
17. Pérez-Jiménez, M.J.; Riscos-Núñez, A. A Linear Solution for the Knapsack Problem Using Active Membranes, in C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa (eds.),*Membrane Computing*. Lecture Notes in Computer Science, **2933**, 2004, 250–268.
18. Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F. A Polynomial Complexity Class in P systems Using Membrane Division, in E. Csuhaj–Varjú, C. Kintala, D. Wotschke, and Gy. Vaszyl (eds.), *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems*, Budapest, Hungary, 2003, 284–294.
19. Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F. Solving VALIDITY Problem by Active Membranes with Input, in M. Cavaliere, C. Martín-Vide, Gh. Păun (eds), *Proceedings of the Brainstorming Week on Membrane Computing*, Tarragona, Spain, 2003, Report RGML 26/03, 279–290.
20. Salomaa, A. *Formal Languages*, Academic Press, New York, 1973.
21. `http://www.p-lingua.org`
22. `http://www.p-lingua.org/mecosim/`
23. `http://sourceforge.net/p/pmcgpu/`

On Parallel Array P Systems

Linqiang Pan[1], Gheorghe Păun[2]

[1] Key Laboratory of Image Processing and Intelligent Control
School of Automation
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
gpaun@us.es, curteadelaarges@gmail.com

Summary. We further investigate the parallel array P systems recently introduced by K.G. Subramanian, P. Isawasan, I. Venkat, and L. Pan. We first make explicit several classes of parallel array P systems (with one or more axioms, with total or maximal parallelism, with rules of various types). In this context, some results from the above mentioned paper by Subramanian et al. are improved. A series of open problems are formulated.

1 Introduction

The generality/versatility of membrane computing is already a well known fact, the computing framework abstracted from the cell structure and functioning can cover a large variety of processes, dealing – in particular – with a large variety of objects processed in the compartments of membrane structures. The arrays (in general, 2D and 3D figures of various types) are one of the types of objects considered already since 2001, see [3]. A direct extension from string objects to two-dimensional arrays was introduced in [1] and then investigated in a series of papers.

A recent contribution to this research area is [6], where a natural counterpart of the array P systems from [1] is considered: parallel rewriting of arrays, instead of the sequential rewriting from [1]. Actually, the kind of parallelism investigated in [6] is that suggested by Lindenmayer systems: all nonterminals of an array should be rewritten in each step. A possible alternative, closer to the style of membrane computing, is to consider the maximal parallelism: a multiset of rules is used which is maximal among the multisets of applicable rules in a given moment.

In the present paper, we explicitly consider these two kinds of parallelism, and we prove that most of the results from [6] hold true for both kinds of parallelism,

also improving those results (less membranes are used in some of them, while the powerful priority relation is avoided in other results).

Several questions remain open; several topics for further research are formulated.

2 Definitions and Notations

It is useful for the reader to be familiar with basic elements of membrane computing, e.g., from [4] (with up-dated information available at [7]), and of array grammars, but the used notions will be recalled below. Actually, in what concerns the arrays, we will usually use the pictorial representation, hence we need a minimal formalism (otherwise, cumbersome if rigorously formulated).

The arrays we consider consist of finitely many symbols from a specified alphabet V placed in the points (we call them *pixels*) of $\mathbf{Z}^2$ (the plane); the points of the plane which are not marked with elements of V are supposed to be marked with the *blank symbol* $\# \notin V$. Given an array W over V, $supp(W)$ denotes the set of points in $\mathbf{Z}^2$ marked with symbols in V. In order to specify an array, it is usual to specify the pixels of the support, by giving their coordinates, together with their associated symbols from V, but, as we said above, we will pictorially represent the arrays, indicating their non-blank pixels. These pictures should be interpreted as arrays placed in any position of the plane (congruent, possible to be superposed by means of a translation).

We denote by V^{*2} the set of all two-dimensional arrays of finite support over V, including the empty array, denoted by λ. Any subset of V^{*2} is called an *array language*.

We handle the arrays by means of rewriting rules. An *array rewriting rule* (over an alphabet V) is written as a usual string rewriting rule, in the form $W_1 \to W_2$, where W_1, W_2 are *isotonic* arrays over V: W_1 and W_2 cover the same pixels, no matter whether they are marked with symbols in V or with #. When graphically representing an array, usually we ignore the blank pixels, but, when representing rewriting rules, the pixels marked with # are also explicitly shown. A rule as above is used to rewrite an array W in the natural way: a position in W is identified where W_1 can be superposed, with all pixels matching, whether or not they are marked with symbols in V or with #, and then those pixels are replaced with W_2 (the fact that W_1 and W_2 are isotonic ensures the fact that this replacement is possible). If the result is the array W', we write $W \Longrightarrow W'$. The reflexive and transitive closure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

Similar to string rewriting rules, the array productions can be classified according to their form. We consider here only two types of rules, *context-free* and *regular*. Remember that all rules we work with are isotonic (the shapes of the left hand side and the right hand side are identical, only the marking, by blank or non-blank symbols, differs). Thus, a context-free rule is an isotonic one with only one non-blank pixel in its left hand side.

Note that we have not distinguished between terminal and nonterminal symbols, like in Chomsky grammars; for regular rules we need such a distinction. Thus, a regular rule over the alphabets T and N, N being the nonterminal one, is a rule of one of the following forms: $A\ \# \to a\ B,\ \#\ A \to B\ a,\ \begin{matrix}\# \\ A\end{matrix} \to \begin{matrix}B \\ a\end{matrix},\ \begin{matrix}A \\ \#\end{matrix} \to \begin{matrix}a \\ B\end{matrix},\ A \to B,\ A \to a$, where $A, B \in N$ and $a \in T$.

Because in what follows we work, like in [6], in a Chomsky framework, with terminal and nonterminal symbols, we also impose that in a context-free rule the single non-blank pixel in the left hand side is marked with a nonterminal symbol.

Before introducing the array P systems, we recall a notion useful below: *two-dimensional right-linear grammars*.

Such a grammar [2] is a construct $G = (V_h, V_v, V_i, T, S, R_h, R_v)$, where V_h, V_v, V_i are the horizontal, vertical, and intermediate alphabets of nonterminals, $V_i \subseteq V_v$, T is the terminal alphabet, $S \in V_h$ is the axiom, R_h is the finite set of horizontal rules, of the forms $X \to AY, X \to A$, for $X, Y \in V_h, A \in V_i$, and R_v is the finite set of vertical rules, of the forms $A \to aB, A \to a$, for $A, B \in V_v, a \in T$.

A derivation in G has two phases, an horizontal one, which uses rules from R_h, and a vertical one, which uses rules from R_v. The horizontal derivation is as usual in a string grammar. In the vertical phase, the rules are used in parallel, downwards, with the restriction that the terminal rules are used simultaneously for all vertical nonterminals. Thus, in the end, a rectangle is obtained, filled with symbols in T. The set of all rectangles generated in this way by G is denoted by $L(G)$ and the family of all languages of this form is denoted by $2RLG$.

Two array languages which will be used below are L_R, of all hollow rectangles with the edges marked with a (one element of this language is shown in Fig. 1), and L_S, of all hollow squares with the edges marked with a.

$$\begin{array}{ccccccccc} a & a & a & a & a & a & a & a & a \\ a & & & & & & & & a \\ a & & & & & & & & a \\ a & & & & & & & & a \\ a & a & a & a & a & a & a & a & a \end{array}$$

Fig. 1. A hollow rectangle in L_R.

3 Parallel Array P Systems

We pass now to define the *parallel array P systems*. Such a device (of degree $m \geq 1$) is a construct

$$\Pi = (V, T, \#, \mu, F_1, \ldots, F_m, R_1, \ldots, R_m, i_o),$$

where: V is the total alphabet, $T \subseteq V$ is the terminal alphabet, $\#$ is the blank symbol, μ is a membrane structure with m membranes labeled in a one-to-one way

with $1, 2, \ldots, m$, $F_1, \ldots, F_m$ are finite sets of arrays over V associated with the m regions of μ, $R_1, \ldots, R_m$ are finite sets of array rewriting rules over V associated with the m regions of μ; the rules have attached targets *here, out, in* (in general, *here* is omitted), hence they are of the form $W_1 \to W_2(tar)$; finally, i_o is the label of a membrane of μ specifying the output region.

In what follows, we only consider array P systems with regular (REG) and context-free (CF) rules – with the symbols in $V - T$ considered as nonterminals.

A computation in an array P system is defined in the same way as in a symbol object P system, with the following details. Each array from a compartment of the system must be rewritten by the rules in that compartment. The rewriting is parallel, with two types of parallelism: (1) *the total one*, indicated by *allP*, which means that all nonterminal symbols from the array are rewritten, and (2) *the maximal one*, indicated by *maxP*, which means that a multiset of rules is applied which is maximal, no further rule can be added to it. For any two rules used simultaneously, no pixel of their left hand sides may overlap (i.e., cover the same pixel of the rewritten array).

An important point appears here in what concerns the target indications of the rules: in each compartment, in a step we apply a multiset of rules with the same target indication. This is a very strong restriction, because it refers to all arrays from the compartment. In this paper, we work under this restriction. A weaker and somewhat more natural condition, which remains to be investigated (e.g., are the results proved below valid also in this case?), is to impose the restriction to use rules with the same target separately for each rewritten array (thus, separate arrays may be rewritten by rules with different targets). Of course, the two variants coincide for systems with only one axiom in the initial configuration.

The arrays obtained by an *allP* or a *maxP* rewriting are placed in the region indicated by the target associated with the used rules, in the usual way in membrane computing. It is important to stress the fact that all arrays from a given compartment travel together during a computation.

A computation is successful only if it halts; that is, it reaches a configuration where no rule can be applied to the existing arrays. The result of a halting computation consists of the arrays composed only of symbols from T placed in the region with label i_o in the halting configuration. The set of all such arrays computed (we also say *generated*) by a system Π is denoted by $A(\Pi)$.

Note that a computation which produces a terminal array (hence no rule can be applied to it), but still can rewrite another array, is not halting; if the rewriting of one array continues forever, no matter how many terminal arrays were produced, then no result is obtained.

We denote by $PAP_m(ax_k, \alpha, \beta)$ the family of all array languages $A(\Pi)$ generated by systems Π as above, with at most m membranes, at most k initial arrays in its compartments ($\Sigma_{i=1}^{m} card(F_i) \leq k$), with rules of type $\alpha \in \{REG, CF\}$, working in the $\beta \in \{allP, maxP\}$ mode. When m or k is not bounded, then it is replaced with $*$.

The following results were proved in [6] (*pri* indicates the use of a priority relation on the rules):

Lemma 1 (Lemma 3 in [6]). $2RLG \subseteq PAP_3(ax_1, CF, allP)$.

Lemma 2 (Lemma 4 in [6]). $PAP_3(ax_1, CF, allP) - 2RLG \neq \emptyset$.

Lemma 3 (Theorem 3 in [6]). $L_R \in PAP_2(ax_1, REG, allP, pri)$.

Lemma 4 (Theorem 4 in [6]). $L_S \in PAP_3(ax_1, REG, allP, pri)$.

In what follows, we will improve all these results in terms of the number of membranes in the first two lemmas and avoiding the priority relation in the last two lemmas (these two results are obtained at the price of using more than one axioms or using the *maxP* way of applying the rules).

4 Results

The first two lemmas above can be easily improved.

Proposition 1. $2RLG \subseteq PAP_2(ax_1, CF, \beta), \beta \in \{allP, maxP\}$.

Proof. Let $G = (V_h, V_v, V_i, T, S, T_h, R_v)$ be a two-dimensional right-linear grammar. We construct the array P system Π indicated in Figure 2. The horizontal derivation is done in membrane 2. When the horizontal phase is completed, the array is moved to the skin membrane, where the vertical phase is performed. After the use of terminal rules, the array is moved back into the inner membrane. If it is not terminal, then the computation continues forever, by means of the rules $A \to A, A \in V_i$. These rules are also introduced in order to make the *maxP* computations to be *allP* computations. The equality $L(G) = A(\Pi)$ is clear. □

Proposition 2. $PAP_2(ax_1, CF, \beta) - 2RLG \neq \emptyset,\ \beta \in \{allP, maxP\}$.

Proof. Let us consider the array P system from Figure 3. From the axiom AB, we generate a string X^nY^n (A goes to the left, simultaneously with B going to the right), then, like in a two-dimensional right-linear grammar, in the skin region we go vertically, marking the pixels with a in the columns of X and with b in the columns of Y. We obtain (moved in the central membrane) a rectangle with the same number of columns marked with a and with b, which is not in the family $2RLG$ (the same example was used also in [6]). The system works identically in the *allP* and *maxP* modes. □

Removing the priority from the other two results from [6] can be done, but making use of the possibility of having two axioms in the initial configuration of the systems. One of them will generate the desired arrays, the other one will generate "twin" arrays, which control the computation of the former arrays.

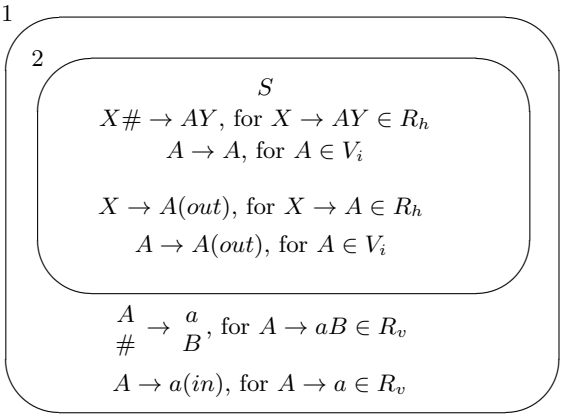


Fig. 2. The array P system from the proof of Proposition 1

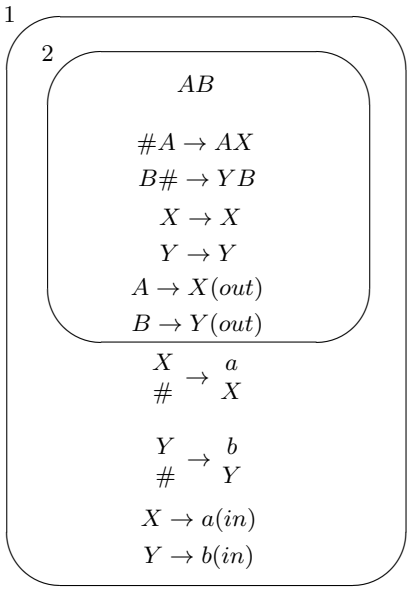


Fig. 3. The array P system from the proof of Proposition 2

Proposition 3. $L_R \in PAP_2(ax_2, REG, \beta),\ \beta \in \{allP, maxP\}$.

Proof. We consider the array P system from Figure 4. The axioms and the rules are written on two columns, in the left one those which lead to the desired arrays, in the right one those handling the "twin" (control) arrays. The rules are similar, with the right column ones dealing with primed nonterminal symbols.

The axiom in the left column is placed in the skin region, the one in the right column is placed in the inner membrane. In the first step, we move the inner axiom to the skin membrane, while removing the subscript 0 from all symbols A, B, A', B'. Now we start the generation of the hollow rectangles.

The bottom horizontal line of the arrays is generated in the skin membrane, by moving B and B' to the right. At a given step, we switch to moving vertically, with the arrays moved to membrane 2. We go upwards, synchronously, then the arrays are again moved to the skin membrane, with D and D' being the current nonterminals. Both D and D' go to left and, at some moment, D is replaced with a (hence the "left" array is terminal (but we do not know whether the rectangle is completed). Moved again in membrane 2, the left array remains idle, while the right one can be rewritten – and, because of the parallelism, this must be done – if (and only if) D' has a non-marked pixel in its left. This happens if and only if the terminal array is not a complete hollow rectangle. The symbol D'' will go up indefinitely, hence the computation never halts. Thus, only the halting computations produce elements in the language L_R. □

A similar result can be obtained for the language of hollow squares.

Proposition 4. $L_S \in PAP_2(ax_2, REG, \beta),\ \beta \in \{allP, maxP\}$.

Proof. We consider the array P system from Figure 5, again with the axioms and the rules written on two columns, with the same significance as above. In the first step, we move the axiom from the skin region to the inner membrane, while also removing the subscript 0.

In the inner membrane we start constructing the squarea, from the left bottom corner, growing here the left and the bottom edges. At some moment, the two arrays are moved to the skin membrane, where the upper and the right edges are constructed. The work on the "left" array is terminated at the moment when the arrays are moved to the inner membrane. We check here whether or not the square is completed. It is not completed if and only if the nonterminal B'' has a non-marked pixel above it. If this is the case, the computation will continue forever, with B''' going to the right, hence the computation never halts. Checking all details remains as an exercise for the reader. □

The *maxP* mode of using the rules is, intuitively speaking, able of "appearance checking", which is known to be a powerful feature of regulated grammars. This is confirmed also in our framework: we can generate the languages L_R, L_S by means of array P systems with only one axiom, at the price of using context-free rules – actually, in the construction below we have only three non-regular rules, of rather simple forms – used in the *maxP* mode.

Proposition 5. $L_R \in PAP_2(ax_1, CF, maxP)$.

Proof. We consider the array P system from Figure 6. We start in the skin membrane, growing the bottom edge. At some moment, we pass to growing the vertical edges, and the array is moved to the inner membrane.

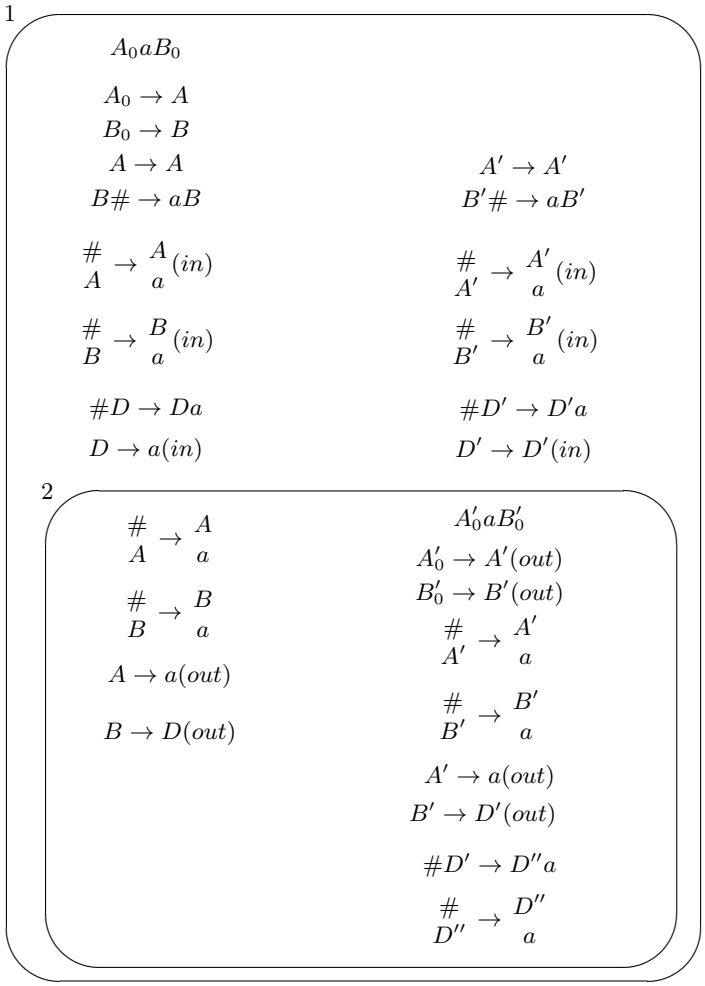


Fig. 4. The array P system from the proof of Proposition 3

After the array is moved to the inner membrane, the symbol B introduces two nonterminals, X and B'. The latter one will grow the right hand edge, the symbol X waits unchanged until the array is moved back to the skin membrane. Here we both grow the upper edge, by means of the nonterminal C, and we change X to Y, which is moved to the left, simultaneously with the symbol C.

At any moment, C is replaced with D and Y with Y' and the array is sent to membrane 2. Here we check whether the rectangle is completed. D is replaced with a. The symbol Y' can be rewritten if and only if it has a non-marked pixel

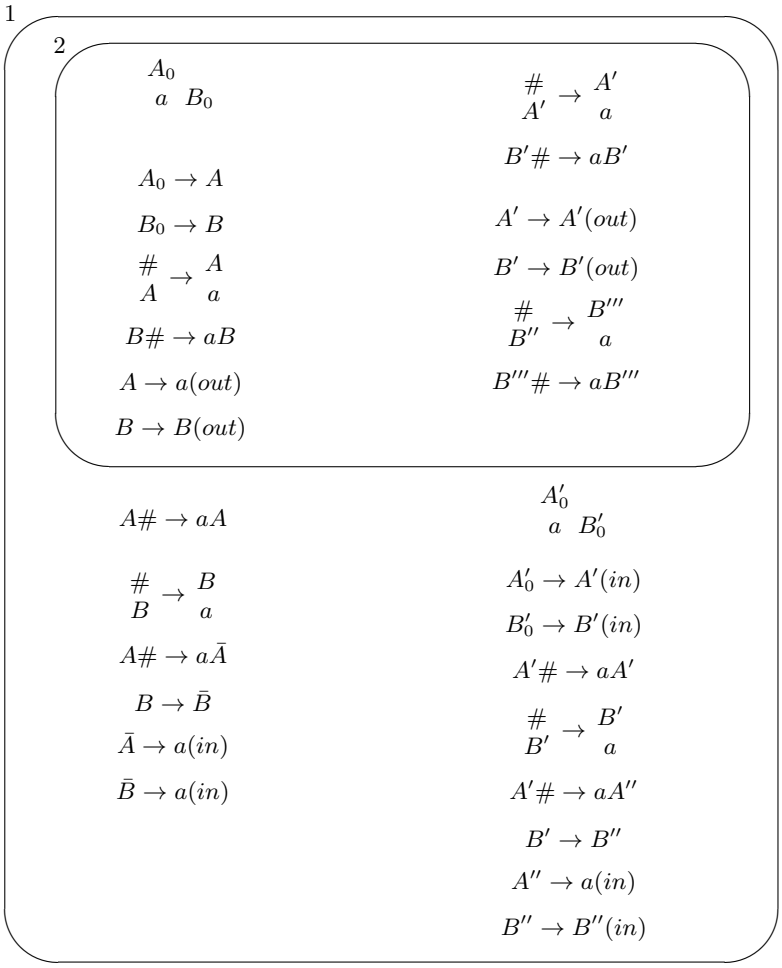


Fig. 5. The array P system from the proof of Proposition 4

in its left hand place. If this is the case, hence the rectangle is not complete, the symbol Z is introduced; if not, the rule $\#Y' \to Z\#(out)$ cannot be applied (the *maxP* mode of using the rules allows rewriting only some nonterminals). Anyway, the array is moved back to the skin region (at least the rule $D \to a(out)$ is used). If Y' is still present, then it is simply erased by the rule $Y' \to \#$. Symbol Z cannot be removed, hence the computation leads to a terminal array if and only if the rectangle is completed. □

It remains as an exercise for the reader to use the same idea in order to prove that $L_S \in PAP_2(ax_1, CF, maxP\}$. On the other hand, we see no way to replace *maxP* with *allP* in these two results: $L_R \in PAP_2(ax_1, CF, maxP)$ and $L_S \in PAP_2(ax_1, CF, maxP)$.

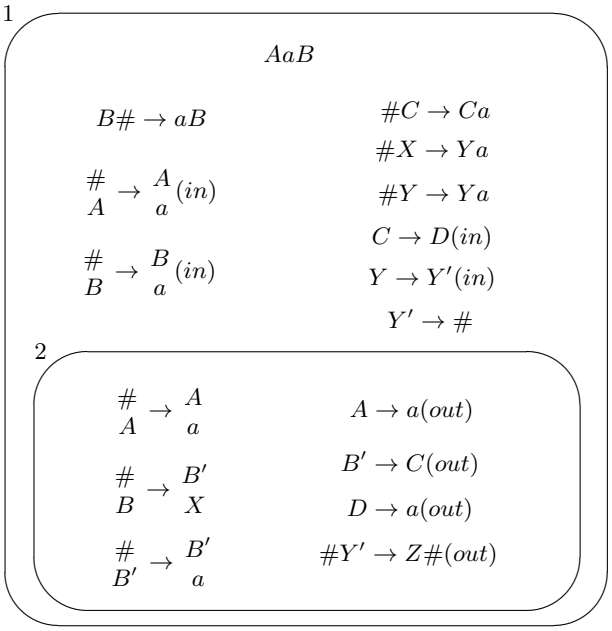


Fig. 6. The array P system from the proof of Proposition 5

5 Concluding Remarks

The goal of this note was, on the one hand, to make explicit the features involved in an array P system (especially, the number of axioms and the two types of parallelism), and to make use of them in order to improve some of the results in [6], in particular, to get rid of the powerful ingredient of the priority relation. Further research efforts are necessary, to clarify the computing power of parallel array P systems. For instance, we have the families $PAP_m(ax_k, \alpha, \beta)$, for $m \geq 1, k = 1$, $\alpha \in \{CF, REG\}, \beta \in \{allP, maxP\}$. What are the relations among them? Do the parameters m and k induce infinite hierarchies? Besides CF and REG one can also consider non-erasing CF rules (no restriction in this respect was considered here).

A natural question is whether or not parallel array P systems are universal. (The sequential ones considered in [1] are universal.)

Rather natural is the possibility, already mentioned, to impose the use of rules with the same target for each array, separately, thus making possible that two arrays from a given region can go in different places after rewriting. A related issue is to consider other ways to control the communication, such as the t mode from grammar systems area, already investigated, for the sequential case, for array P systems in [5].

Finally, we mention the problem of considering Lindenmayer-like array P systems, that is, pure (without nonterminals) or extended (with all symbols to be rewritten, but accepting only terminal arrays).

Aknowledgements. The work of the first author was supported by National Natural Science Foundation of China (61033003, 91130034 and 61320106005).

References

1. R. Ceterchi, M. Mutyam, Gh. Păun, K.G. Subramanian: Array-rewriting P systems. *Natural Computing*, 2 (2003), 229–249.
2. D. Giammarresi, A. Restivo: Two-dimensional languages. In *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), Vol. 3, Springer-Verlag, Berlin, 1997, 215–267.
3. S.N. Krishna, K. Krithivasan, R. Rama: P systems with picture objects. *Acta Cybernetica*, 15, 1 (2001), 53–74.
4. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
5. K.G. Subramanian, R.M. Ali, A.K. Nagar, M. Margenstern: Array P systems and t communication. *Fundamenta Informaticae*, 91, 1 (2009), 145–159.
6. K.G. Subramanian, P. Isawasan, I. Venkat, L. Pan: Parallel array-rewriting P systems. *Romanian Journal of Information Science and Technology*, to appear.
7. The P Systems Website: `http://ppage.psystems.eu`.

Four (Somewhat Nonstandard) Research Topics

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
gpaun@us.es, curteadelaarges@gmail.com

Summary. Four research directions are suggested, dealing with the following four main ideas: computing along the axon (up to now, this topic was only preliminarily investigated), using pre-computed resources in order to solve computationally hard problems, considering in P systems both objects "of matter" and "of anti-matter" (which annihilate each other when meet), and considering the distance (naturally defined in a membrane structure of a given type) as a support of information.

1 Introduction

Membrane computing is more than fifteen years old, [6], and it is already *a well established branch of natural computing* (it is no longer appropriate to call it "a young branch of NC..."), but, in spite of its large bibliography, there are a lot of open problems and research topics in circulation. It is enough to mention the "mega-paper" [2].

Still, further problems and research ideas can be imagined – four classes are suggested here.

Of course, the reader is assumed to be familiar with membrane computing, so that the presentation is minimal, both in what it concerns the details and the references.

2 Computing Along the Axon

The motivation is related to the fact that the axons are not simple "wires" among neurons, but they carry computations, moving spikes back and forth, amplifying them (in the so-called Ranvier nodes) etc. Starting from this, *axon P systems* were introduced in [1] (and then investigated in [8]). The idea is suggested in Figure 1.

We have [Ranvier] nodes arranged along an axon. Initially, each of them can contain a number of spikes. A rule $E/a^c \to (a^l, a^r)$ is applied as in an SN P system;

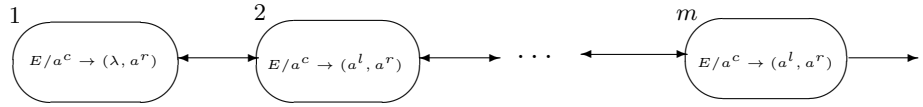


Fig. 1. An axon P system

c spikes are consumed, l spikes are sent to the left node, and r spikes are sent to the right node. The rightmost node is the output one.

The functioning is like that of an SN P system (with a fixed synapses graph and specific spiking rules).

In [1] the axon P systems were used as string/sequence generators: if the output node sends out k spikes, then the symbol b_k is associated with that step of the computation; no spikes means the symbol b_0. In the halting case, only recursive languages can be generated.

Several problems to investigate in this framework are natural (part of them also formulated in the cited papers):

1. generate numbers, by considering the number of steps elapsed between the first two spiking steps (like in SN P systems, halting or not);
2. in the case of strings, look for variants which are computationally universal (what about ignoring the steps when no spike exits, hence taking $b_0 = \lambda$?);
3. consider axon P systems as infinite sequence generators;
4. look for hierarchies (on the number of axons) in the non-universal case;
5. what about also using anti-spikes, like in SN P systems, [5]? (besides the object a, also $\bar{a}$ is considered, called *anti-spike*; spiking rules can consume and/or create either spikes or anti-spikes, but a node/neuron cannot contain at the same time both spikes and anti-spikes, they immediately annihilate themselves, by means of an implicit rule $a\bar{a} \to \lambda$, which is used in no time, in a maximal way).

The axon P systems can be of interest also as a support for the last problem suggested here, i.e., using the distance (between two nodes) as a support of information.

3 Pre-Computed Resources

The idea has been explored in a couple of papers (see, e.g., [3], [4]), but still there is no formal definition of the complexity classes which appear in this context (the problem is, in fact, to define "acceptable" pre-computed resources; intuitively, the idea is to have a pre-computed arbitrarily large support for computation of the form uv^ω, with information only in u; how to pass from this string intuition to SN P systems – or to other types of P systems – remains open; in [7], page 360, it is suggested that the concept of *uniformity* from Circuit Complexity can be useful in this respect).

Besides recalling the attention of the previous problem, I would like to point out here another question which looks of (practical) interest: is it possible to find a pre-computed support of computation which can be used for solving two/several/all problems from a given class? This would be interesting and important: "constructing the computer", whatever large it is, is done only once, then several problems can be solved using the same pre-computed "hardware", where specific information is plug-in.

4 Using Matter and Anti-matter

This is a direct generalization of the idea of anti-spikes from SN P systems, [5]: for each object a, in any type of P systems, to consider an "anti-object" $\bar{a}$. Both objects and anti-objects are handled by usual evolution rules, but matter and anti-matter cannot stay together in a compartment, a rule of the form $a\bar{a} \to \lambda$ is supposed to exist in any place. This rule is applied immediately, in no time, so that in each compartment, in the end of each evolution step either only objects or only anti-objects are present.

Which is the effect, in various classes of P systems, of using anti-objects? In SN P systems, anti-spikes help, so it is expected that this happens also in other cases (and this is one further illustration of the power of annihilation rules $a\bar{a} \to \lambda$, known to be useful in formal language theory, and possibly also in the case of multiset rewriting).

In particular: are one catalytic (purely two catalytic) P systems with anti-objects universal?

The following trivial proof of the universality of P systems with two catalysts and with anti-objects supports the conjecture that the answer to the previous question might be positive.

Formally, we can write $NOP_1(2cat, antim) = NRE$, with the obvious meaning.

Indeed, let us consider a 3 register machine, with register 1 (the one where the result is obtained) never decremented, and with registers 2, 3 empty in the end of a computation. Denote it by $M = (3, H, l_0, l_h, I)$ and construct a one-membrane catalytic P system with two catalysts, c_2, c_3. Initially, the system contains the objects l_0, c_2, c_3 in its membrane. The contents of register r is represented by the number of copies of an object a_r, $r = 1, 2, 3$, in the system. For $r = 2, 3$ we also consider the anti-objects $\bar{a}_r$.

For each instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in $I, r = 1, 2, 3$, we consider the rules

$$l_i \to l_j a_r d_2 d_3,$$
$$l_i \to l_k a_r d_2 d_3.$$

For each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in $I, r = 2, 3$, we consider the rules

$$l_i \to l_j \bar{a_r} d_2 d_3,$$
$$\bar{a_r} \to \#,$$

$$l_i \to l_k d_{5-r},$$
$$c_r d_r \to c_r,$$
$$c_r a_r \to c_r \#.$$

We also add the rules

$$\# \to \#,$$
$$l_h \to \lambda.$$

The rule $l_i \to l_j \bar{a}_r d_2 d_3$ is used when register r is non-empty; the decrement by 1 is done by means of the implicit rule $a_r \bar{a}_r \to \lambda$. If this rule cannot be applied, then the trap object is introduced, by means of $\bar{a}_r \to \#$. If the register is empty, then the rule $l_i \to l_k d_{5-r}$ should be used.

Note the role of objects d_2, d_3, which "keep busy" the catalysts, not to act in rules of the form $c_r a_r \to c_r \#$. For instance, if the rule $l_i \to l_k d_{5-r}$ is used, but register r is not empty, then, because object d_r is not introduced, but only d_{5-r}, in the next step, the rule $c_r a_r \to c_r \#$ should be used, and the computation never stops.

When the computation in M halts, the object l_h is removed. The number of objects a_1 in the system equals the number computed by M. (The catalysts should be ignored, or the results can be read outside the system, sending a_1 out, etc.)

5 Using the Distance to Encode Numbers

The idea is again suggested by the spiking neurons. If we watch the movie from `http://www.igi.tugraz.at/tnatschl/spike_trains_eng.html`, we see that the distance between various spikes moving along an axon is relevant for the process/computation. How to make this explicit in a P system? Of course, in order to can handle arbitrarily large numbers, we need a membrane structure of an arbitrarily large size, hence we need ways for generating space (membranes in a cell-like or tissue-like P system, neurons in an SN P system, nodes in an axon P system), or we have to deal, again, with pre-computed resources.

Define distance-based P systems, and investigate their power and efficiency. Can universality be reached when using only a bounded number of objects? (I expect a positive answer.) Which are the shapes of the membrane structure necessary/sufficient for computing various classes of numbers, or for the conjectured universality.

6 Final Remarks

Some of the previous questions are more precise, others are less precise. Maybe some of them are trivial. Anyway, a brainstorming is the right place to discuss such issues – my big regret not to be there. I would be very much indebted to the reader present in Sevilla for any reaction.

References

1. H. Chen, T.-O. Ishdorj, and Gh. Păun: Computing along the axon. *Proc. 4th BWMC*, 2006, vol. I, 225-240, and *Progress in Natural Science*, 17(4) (2007), 417-423.
2. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Frontiers of membrane computing: Open problems and research topics, *Intern. J. Found. Computer Sci.*, 2013 (first version in *Proc. Tenth Brainstorming Week on Membrane Computing*, Sevilla, January 30 – February 3, 2012, vol. I, 171–249).
3. T.-O. Ishdorj, A. Leporati: Uniform solutions to `SAT` and `3-SAT` by spiking neural P systems with pre-computed resources. *Natural Computing*, 7 (2008), 519–534.
4. A. Leporati, M.A. Gutiérrez-Naranjo: Solving `SUBSET SUM` by spiking neural P systems with pre-computed resources. *Fundamenta Informaticae*, 87 (2008), 61–77.
5. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. *Intern. J. Computers, Comm. Control*, 4, 3 (2009), 273–282.
6. Gh. Păun: Computing with membranes. *J. Comput. Syst. Sci.*, 61 (2000), 108–143 (see also TUCS Report 208, November 1998, `www.tucs.fi`).
7. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
8. X. Zhang, J. Wang, and L. Pan: A note on the generative power of axon p systems. *International Journal of Computers, Communications, and Control*, 4 (2009), 92-98.
9. The P Systems Website: `http://ppage.psystems.eu`.

Membrane Clustering: A Novel Clustering Algorithm under Membrane Computing

Hong Peng[1], Jiarong Zhang[1], Jun Wang[2], Tao Wang[3],
Mario J. Pérez-Jiménez[4], Agustín Riscos-Núñez[4]

[1] Center for Radio Administration and Technology Development,
Xihua University, Chengdu 610039, China
ph.xhu@hotmail.com
[2] School of Electrical and Information Engineering,
Xihua University, Chengdu 610039, China
[3] School of Electrical Engineering,
Southwest Jiaotong University, Chengdu 610031, China
[4] Research Group of Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Seville, Sevilla 41012, Spain

Summary. Membrane computing (known as P systems) is a class of distributed parallel computing models, this paper presents a novel algorithm under membrane computing for solving the data clustering problem, called as membrane clustering algorithm. The clustering algorithm is based on a tissue-like P system with a loop structure of cells. The objects of the cells express the candidate cluster centers and are evolved by the evolution rules. Based on the loop membrane structure, the communication rules realize a local neighborhood topology, which helps the co-evolution of the objects and improves the diversity of objects in the system. The tissue-like P system can effectively search for the optimal clustering partition with the help of its parallel computing advantage. The proposed clustering algorithm is evaluated on four artificial data sets and six real-life data sets. Experimental results show that the proposed clustering algorithm is superior or competitive to classical k-means algorithm and several evolutionary clustering algorithms recently reported in the literature.

1 Introduction

Data clustering is a fundamental conceptual problem in data mining, which describes the process of grouping data into classes or clusters such that the data in each cluster share a high degree of similarity while being very dissimilar to data from other clusters [1]. Over the past years, a large number of clustering algorithms have been proposed [2, 3, 4], which can be divided roughly as two categories: hierarchical and partitional. Hierarchical clustering proceeds successively by either merging smaller clusters into larger ones or by splitting larger clusters. Partitional

clustering attempts to directly decompose a data set into several disjointed clusters based on similarity measure, for example, mean square error (MSE). Clustering algorithms have been used in a wide variety of areas, such as pattern recognition, machine learning, image processing, web mining [5, 6]. In the present study, the classical k-means algorithm [7, 8] has received wide attention because of the following two reasons: (i) k-means has been recently elected and listed among the top most influential data mining algorithms [9]; (ii) it is at the same time very simple and quite scalable, as it has linear asymptotic running time with respect to any variable of the problem. However, k-means is sensitive to the initial cluster centers and easy to get stuck at the local optimal solutions. Moreover, k-means takes large time cost to find the global optimal solution when the number of data points is large.

In recent years, some evolutionary algorithms have been introduced to overcome the shortcomings of k-means algorithm because of their global optimization capability. Several genetic algorithms (GA)-based clustering algorithms were proposed, which used the two different methods to express the clustering solutions respectively. The first method uses the chromosome directly to encode the cluster number that each data point belongs to [27, 28]. However, this method does not reduce the size of the search space and searching cost of the optimal solution when the data points proliferate. Another method uses a relatively indirect representation, in which the chromosome encodes the cluster centers and each data is subsequently assigned to the closest cluster center [10, 11, 12, 13, 14]. However, most of GA-based clustering algorithms can suffer from the degeneracy when numerous chromosomes represent the same solution. The degeneracy can lead to inefficient coverage of the search space as the same configurations of clusters are repeatedly explored. It is for this reason that some researchers developed the particle swarm optimization (PSO)-based or ant colony optimization (ACO)-based clustering algorithms. Kao et al. have proposed a hybrid technique based on combining the k-means and PSO for cluster analysis [15]. Shelokar et al. have introduced an evolutionary algorithm based on ACO for clustering problem [16]. Niknam et al. have presented a hybrid evolutionary optimization algorithm based on the combination of PSO and ACO for solving the clustering problem [17].

Membrane computing initiated by Păun [18] in 1998, is inspired by the structure and functioning of living cells as well as the interaction of living cells in tissues, organs or neural nets. Membrane computing is a novel class of distributed parallel computing models, and also known as P systems. The computing models usually have three key elements: membrane structure, multisets of objects and rules [19]. Generally, the multisets of objects are placed in compartments surrounded by membranes and evolved by some given rules. In recent years, a large number of variants have been proposed [20, 21, 22, 23]. These efforts have addressed the parallel computing advantage of P systems as well as the high effectiveness of solving a variety of difficult problems, especially, P systems can solve a number of NP-hard problems in linear or polynomial time complexity [24]. Moreover, membrane algorithms, as a variant of P systems, have demonstrated a powerful global

optimization performance [25, 26]. This paper focuses on application of membrane computing to data clustering. Our motivation is applying the specially designed elements and inherent mechanisms of P systems to achieve a novel clustering algorithm, called membrane clustering algorithm in this paper.

The rest of this paper is organized as follows. Section 2 gives a brief outline of tissue-like P systems. The proposed membrane clustering algorithm is presented in Section 3, and experimental results and analysis are provided in Section 4. Finally, Section 5 draws the conclusions.

2 Tissue-like P systems

The P systems first proposed are cell-like P systems in which the membranes are arranged as a rooted tree [18, 19], where the root expresses the skin of the cells (the outermost membrane) and the leafs represent elementary membranes (which do not contain any other membrane). Its biological inspiration is from the morphology of the cells, where small vesicles are surrounded by the large vesicles. For tissue-like P systems, tree-like structure is changed as a general graph. It is from the two biological inspirations: intercellular communication and collaboration between neurons. The intercellular communication is based on symport/antiport rules, which are introduced as the communication rules of tissue-like P systems. In symport rules, objects cooperate to traverse a membrane together in the same direction, whereas in the case of antiport rules, objects residing at both sides of the membrane cross it simultaneously but in opposite directions.

Formally, a tissue-like P system (of degree $q > 0$) with symport/antiport rules is a construct

$$\Pi = (O, w_1, \ldots, w_q, R_1, \ldots, R_q, R', i_0) \tag{1}$$

where

(1) O is a finite alphabet, whose symbols are called objects;
(2) $w_i (1 \leq i \leq q)$ is finite set of strings over O, which represents multiset of objects initially present in cell i;
(3) $R_i (1 \leq i \leq q)$ is finite set of evolution rules in cell i;
(4) R' is finite set of communication rules of the form $(i, u/v, j)$, which represents communication rule between cell i and cell j, $i \neq j$, $i, j = 1, 2, \ldots, q$, $u, v \in O^*$;
(5) i_0 indicates the output region of the system.

From membrane structure, a tissue-like P system can be viewed as a net implicitly, which consists of the q cells labeled by $1, 2, \ldots, q$ respectively. Here, each cell is an elementary membrane. Usually, the environment is labeled by 0. The communication rule of the form $(i, u/v, j)$ indirectly indicates synaptic connection between cell i and cell j. 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. The net structure provides the flexibility of expressing the needed structures from simple to complex when we deal with real-world problems.

In tissue-like P systems, multisets of objects of the q cells are described by $w_1, w_1, \ldots, w_q$. Suppose any multiset of objects over O is available in the environment.

Generally speaking, a tissue-like P system includes the rules of two types: evolution rules and communication rules. Each cell usually contains one or more evolution rules, while a communication rule is built between two different cells. In above definition, $R_i(1 \leq i \leq q)$ is finite set of evolution rules in cell i, whose rule is of the form $u \rightarrow v, u, v \in O^*$. The application of the rule means that u will be evolved to v. In most of the existing tissue-like P systems and variants, evolution rule of the form is based on string of objects. However, when we apply it to solve real-world problem, we should design the corresponding evolution rules according to domain knowledge of the real-world problem. The communication rule of the form $(i, u/v, j)$ is called as antiport rule. The communication rule $(i, u/v, j)$ can be applied over two cells labeled by i and j when 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. Note that if either $i = 0$ or $j = 0$ then the objects are interchanged between a cell and the environment. If one of u or v in above rule is empty, the rule is called as symport rule, for example, $(i, u/\lambda, j)$. The application of the rule means that u will be communicated form cell i to cell j.

In tissue-like P systems, as usual in the framework of membrane computing, every cell as a computing unit works in a maximally parallel way (a universal clock is considered here). In a computing step, each object in a cell can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we can apply a maximal set of rules.

A computation in a tissue-like P system of degree d is a sequence of steps which start with the cells $1, \ldots, q$ containing the multisets $w_1, \ldots, w_q$ and where, in each step, one or more rules are applied to the current multisets of symbol objects. A computation is successful if and only if it halts. When it halts, it produces a final result in output cell.

3 The proposed membrane clustering algorithm

In this section, we will present in detail the developed membrane clustering algorithm, a novel clustering algorithm under the framework of membrane computing, which is based on a tissue-like P system with a loop structure of cells. As usual, the designed tissue-like P system consists of several cells, each of which contains a object or multiple objects. The cells have some evolution rules to evolve the objects of the system, while communication rules between cell membranes are used to exchange and share the objects. Moreover, the loop structure of cells is indicated indirectly by the communication rules. The tissue-like P system can realize the co-evolution of objects among the cells under the control of evolution rules

and communication rules. The role of the tissue-like P system is to search for the optimal cluster centers for a data set to be clustered.

In the following, we first describe several basic components, and then provide the proposed tissue-like P system and membrane clustering algorithm.

3.1 Clustering measure

Suppose that data set D has n sample points, $x_1, x_2, \ldots, x_n$, $x_i \in R^d (i = 1, 2, \ldots, n)$, and is partitioned into k clusters, $C_1, C_2, \ldots, C_k$. Denote by $z_1, z_2, \ldots, z_k$ the corresponding cluster centers. If the distances of sample point x_i to cluster centers $z_p (p = 1, 2, \ldots, k)$ satisfy

$$||x_i - z_j|| \leq ||x_i - z_p||, p = 1, 2, \ldots, k \text{ and } j \neq p, \tag{2}$$

then sample point x_i is assigned to cluster C_j, $i = 1, 2, \ldots, n$.

Usually, partitional clustering algorithm searches for the optimal cluster centers in the solution space according to some clustering measure in order to solve data clustering problem. A commonly used clustering measure is

$$M(C_1, C_2, \ldots, C_k) = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - z_i||. \tag{3}$$

Generally, the smaller the M value, the higher the clustering quality. In this work, the clustering measure is also used to evaluate the objects of the system during object evolution. If the M value of an object is the smaller, the object is the better, otherwise, it is worse.

3.2 Membrane structure

The membrane clustering algorithm proposed in this paper is based on a tissue-like P system of degree q, which consists of q cells, shown in Fig. 1. The cells are labeled by $1, 2, \ldots, q$, respectively. The region labeled by 0 is the environment. In this work, the environment is also output region of the system. The directed lines in Fig. 1 indicate the communication of objects between the q cells. Moreover, the q cells will be arranged as a loop topology based on the communication rules described below. As usual in P system, the q cells, as parallel computing units, will run independently. In addition, the environment always stores the best object found so far in the system. When the system halts, the object in the environment will be regarded as the output of whole system.

3.3 Objects

In the tissue-like P system, each cell contains several objects. The role of the designed tissue-like P system is to find the optimal cluster centers for a data set,

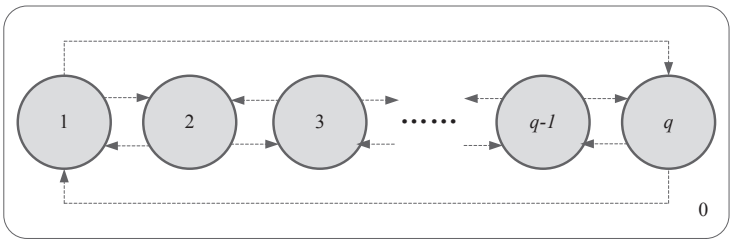


Fig. 1. Membrane structure of the designed tissue-like P system.

thus each object in cells will express a group of (candidate) cluster centers. Since data set D has k cluster centers and each cluster center is a d-dimensional vector, each object in the system is considered as a $(k \times d)$-dimensional real vector of the form

$$\mathbf{z} = (z_{11}, z_{12}, \ldots, z_{1d}, \ldots, z_{i1}, z_{i2}, \ldots, z_{id}, \ldots, z_{k1}, z_{k2}, \ldots, z_{kd})$$

where $z_{i1}, z_{i2}, \ldots, z_{id}$ are d components of ith cluster center z_i, $i = 1, 2, \ldots, k$. For simplicity, suppose that each cell has the same number of objects, which is denoted by m.

Initially, the system will randomly generates m initial objects for each cell. When an initial object $\mathbf{z}$ is generated, $(k \times d)$ random real numbers are produced repeatedly to form it with the constraint of

$$A_1 \leq z_{i1} \leq B_1, \ldots, A_j \leq z_{ij} \leq B_j, \ldots, A_d \leq z_{id} \leq B_d \tag{4}$$

where A_j and B_j are lower bound and upper bound of jth dimensional component of data points, respectively, $j = 1, 2, \ldots, d$.

3.4 Rules

The tissue-like P system includes the rules of two types: the evolution rules, which aim to evolve the objects in cells and the communication rules, which aim to exchange and share the objects. Evolution rules are used to evolve the objects associated with cluster centers, so the tissue-like P system is able to find the optimal cluster centers for a data set via the evolution of objects. Moreover, communication rules will realize the exchange and sharing of better objects between adjacent cells. Note that in each computing step, the communication rules are executed after the evolution rules. For each cell, the better objects communicated from its two adjacent cells form a subset of objects, called external pool, whose objects will participate its evolution of objects in next computing step (see Fig. 2). As usual in P systems, each cell as an independent computing unit runs in maximum parallel way under the control of a global clock.

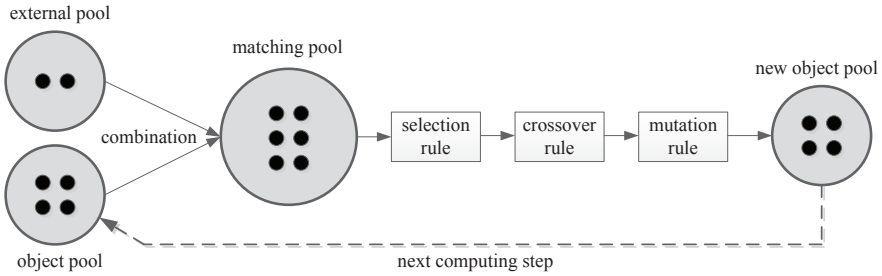


Fig. 2. The evolution procedure of objects in a cell.

Evolution rules

The role of evolution rules is to evolve the objects in cells to generate new objects used in next computing step. During the evolution, each cell maintains the same size (the number of objects). In this work, three known genetic operations (selection, crossover and mutation) [29, 30] are introduced as the evolution rules in cells. In a computing step, all objects (located in object pool) in each cell and the better objects (located in external pool) from its two adjacent cells constitute a matching pool. The objects in external pool are actually the better objects communicated from its two adjacent cells in previous computing step. The objects in matching pool will be evolved by executing selection, crossover and mutation operations in turn. In order to maintain the size of objects in each cell, truncation operation is used to constitute new object pool according to the M values of objects. The objects in new object pool will be regarded as the objects to be evolved in next computing step. Fig. 2 shows the evolution procedure of objects in a cell.

In this work, selection operation uses usual rotating wheel method, while crossover operation uses single-point crossover in which the position of crossover point is determined according to crossover probability p_c [31]. The single-point mutation is used to realize the mutations of objects. If v is a mutation point determined according to mutation probability p_m, its value becomes, after mutating,

$$v' = \begin{cases} v \pm 2 \times \delta \times v, & v \neq 0 \\ v \pm 2 \times \delta, & v = 0 \end{cases} \tag{5}$$

where the signs "+" or "-" occur with equal probability, and δ is a real number in the range [0,1], generated with uniform distribution.

Communication rules

The communication rules are used to exchange the objects between each cell and its two adjacent cells and update the best object found so far in the environment. The tissue-like P system designed in this paper involves the communication rules of two types:

(1) Antiport communication rule: $(i, \mathbf{z}/\mathbf{z}', j)$, $i, j = 1, 2, \ldots, q$. The rule indicates that object $\mathbf{z}$ is communicated from cell i to cell j and object $\mathbf{z}'$ is communicated from cell j to cell i.
(2) Symport communication rule: $(i, \mathbf{z}/\lambda, 0)$, $i = 1, 2, \ldots, q$. The rule expresses that object $\mathbf{z}$ is communicated from cell i to the environment.

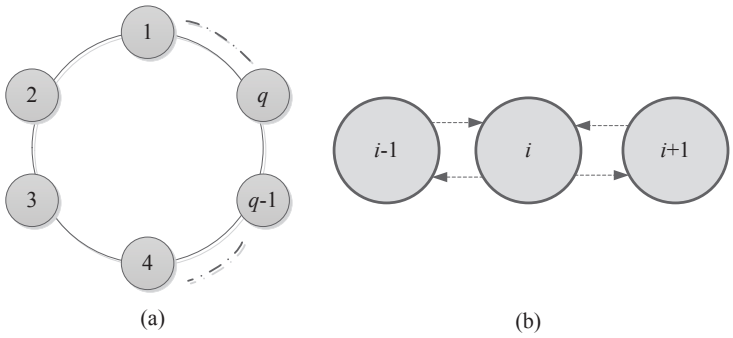


Fig. 3. A loop topology structure of cells and the communication relation between adjacent cells.

The communication rules impliedly indicate the connection relationship between cells. Fig. 3 shows the communication relation of objects between cells in the designed tissue-like P system. From a logical point of view, the communication relation shows that the cells form a loop topology, shown in Fig. 3(a). Meanwhile, this also reflects a neighborhood structure of the communication of objects, namely, each cell only exchanges and shares the objects with its two adjacent cells, shown in Fig. 3(b). After the objects are evolved, each cell (such as cell i) transmits its several best objects into adjacent cells (such as cells $i-1$ and $i+1$) and retrieves several best objects from adjacent cells (such as cells $i-1$ and $i+1$) by using the communication rule, constituting the matching pool of objects in next computing step. The special logical structure can bring the following benefits:

(1) The co-evolution of objects in the q cells can accelerate the convergence of the proposed clustering algorithm.
(2) The object sharing mechanism of the local neighborhood structure can enhance the diversity of objects in the entire system.

The communication of objects not only occurs between cells, but also appears between cell and the environment. The global best object found so far in whole system is stored always in the environment. After objects are evolved, each cell communicates its best object found in current computing step into the environment to update the global best object. The update strategy used in the tissue-like P

system is that if the communicated object is better than the global best object, the global best object is substituted, otherwise it is discarded.

3.5 Halt condition

In this paper, maximum execution step number is used as the halt condition of the tissue-like P system, that is, the tissue-like P system will continue to run until it reaches the maximum execution step number. When the system halts, the best object in the environment is regarded as the system output, which is the found optimal cluster centers.

3.6 The proposed clustering algorithm

According to the components discussed above, the designed tissue-like P system can be formally described as follows. It is a tissue-like P system of degree q,

$$\Pi = (Z_1, \ldots, Z_q, R_1, \ldots, R_q, R', i_o)$$

where

(1) Z_i is the set of m objects in cell i, where each object $\mathbf{z}$ is a $(k \times d)$-dimensional vector, $1 \leq i \leq q$;
(2) R_i is the finite set of evolution rules, $1 \leq i \leq q$. Each R_i contains three evolution rules: selection, crossover and mutation rules;
(3) R' is the finite set of communication rules with the following forms:
   (a) Antiport communication rule, $(i, \mathbf{z}/\mathbf{z}', j)$, $i, j = 1, 2, \ldots, q$, $i \neq j$. The rule is used to communicate the objects between an cell and its two adjacent cells;
   (b) Symport communication rule, $(i, \mathbf{z}/\lambda, 0)$, $i = 1, 2, \ldots, q$. The rule is used to communicate the objects between cell and the environment.
(4) $i_o = 0$ indicates that the environment is the output region of whole system.

Based on the tissue-like P system, the proposed membrane clustering algorithm is summarized in Table 1.

4 Experiment results and analysis

In this section, the proposed membrane clustering algorithm is evaluated on ten data sets and compared with classical k-means algorithm and several clustering algorithms based on evolutionary algorithms, including GA [10], PSO [15] and ACO [16]. In order to test the robustness of these clustering algorithms, we repeat the experiments 50 times for each data set.

Table 1. Membrane clustering algorithm: a clustering algorithm based on tissue-like P systems

Input parameters: Data set, D, the number of clusters, k, the number of cell, q, the number of objects in each cell, m, maximum execution step number, S_{max}, crossover rate, p_c, and mutation rate, p_m.
Output results: the optimal cluster centres, G.
Step 1. **Initialization**
for i=1 *to* q
for j=1 *to* m
Generate jth initial object for cell i, Z_{ij};
Partition all data points into clusters $C_1, C_2, \ldots, C_k$;
Compute the M value of the object, M_{ij};
end for
end for
Fill the global best object G using the best of all initial objects;
Set computing step $s = 0$;
Step 2. **Object evolution in cells**
for each cell i $(i = 1, 2, \ldots, q)$ in parallel *do*
Evolve all object Z_{ij} $(j = 1, 2, \ldots)$ in its mating pool using evolution rules;
Use truncation operation to maintain its m best objects;
for $j = 1$ *to* m
Partition all data points into clusters $C_1, C_2, \ldots, C_k$;
Compute the M value of the object, M_{ij};
end for
end for
Step 3. **Object communication between cells**
for each cell i $(i = 1, 2, \ldots, q)$ in parallel *do*
Transmit better objects in cell i to its two adjacent cells;
Receive better objects from its two adjacent cells into its mating pool;
Update G using the best object in cell i;
end for
Step 4. **Halt condition judgment**
if $s \leq S_{max}$ is satisfied
$s = s + 1$;
goto **Step 2**;
end if
The system exports the global best object G in the environment and halts;

4.1 Data sets

In the experiments, two kinds of data are used to evaluate these clustering algorithms. The first is the four manually-generated data sets used in the existing literatures, *AD_5_2*, *Data_9_2*, *Square_4* and *Sym_3_22*, shown in Fig. 4. The second is the six real-life data sets provided in UCI [32], including the *Iris*, *BreastCancer*, *Newthyroid*, *LungCancer*, *Wine* and *LiveDisorder*.

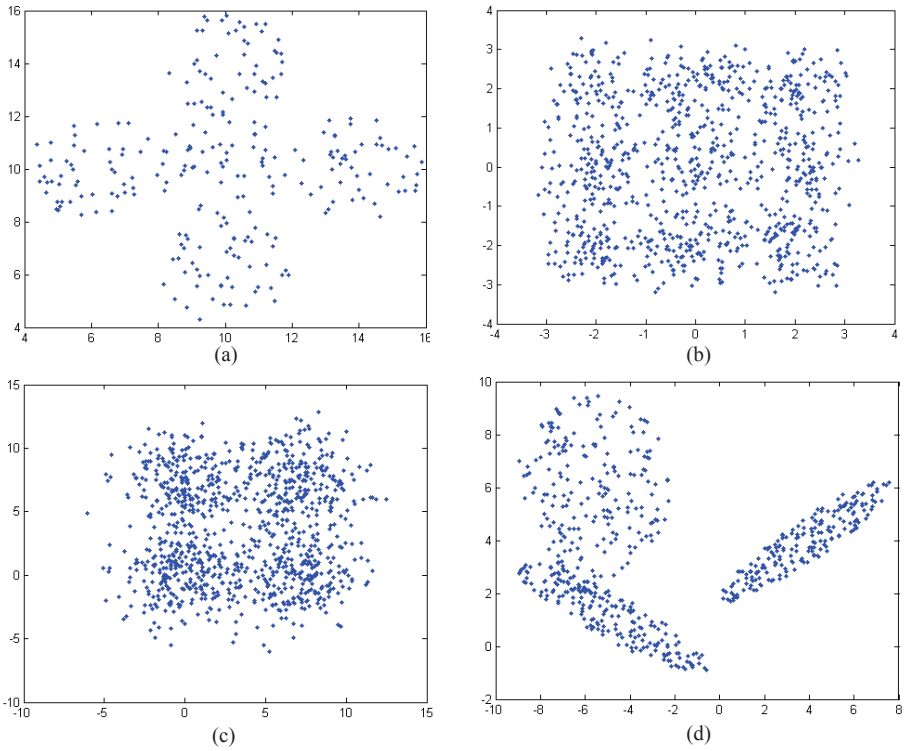


Fig. 4. Four artificial data sets: (a) *AD_5_2*; (b) *Data_9_2*; (c) *Square_4*; (d) *Sym_3_22*.

- *AD_5_2*. This data set consists of 250 two-dimensional data points distributed over five spherically shaped clusters. The clusters present in this data set are highly overlapping, each consisting of 50 data points. This data set is shown in Fig. 4(a).
- *Data_9_2*. This data set consists of 900 two-dimensional data points distributed over nine spherically shaped clusters. The clusters present in this data set are highly overlapping. This data set is shown in Fig. 4(b).
- *Square_4*. This data set consists of 1000 data points distributed over four squared clusters. This data set is shown in Fig. 4(c).

- *Sym_3_22*. This data set consists of 600 two-dimensional data points distributed over three clusters, where first and second clusters are spherically shaped while third cluster is elliptically shaped, each consisting of 200 data points. This data set is shown in Fig. 4(d).
- *Iris*. This data set consists of 150 data points distributed over three clusters. Each cluster consists of 50 points. This data set represents different categories of irises characterized by four feature values in centimeters: the sepal length, sepal width, petal length and the petal width [33]. This data set has three classes, namely, Setosa, Versicolor and Virginica, among which the last two classes have a large amount of overlap while the first class is linearly separable.
- *BreastCancer*. This data set consists of 683 sample points. Each pattern has nine features corresponding to clump thickness, cell size uniformity, cell shape uniformity, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli and mitoses. There are two categories in the data: malignant and benign. The two classes are known to be linearly separable.
- *Newthyroid*. The original database from where it has been collected is titled as thyroid gland data ("normal", "hypo" and "hyper" functioning). Five laboratory tests are used to predict whether a patient's thyroid belongs to the class euthyroidism, hypothyroidism or hyperthyroidism. There are a total of 215 instances and the number of attributes is five.
- *LungCancer*. The data consists of 32 instances having 56 features each. The data describes three types of pathological lung cancers.
- *Wine*: This is a wine recognition data consisting of 178 instances with 13 features resulting from a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.
- *LiveDisorder*. This data set contains 345 instances with six features each. The data has two categories. The first five variables are all blood tests, which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption.

4.2 Setup

In the experiments, the proposed membrane clustering algorithm will be compared with k-means and three evolutionary clustering algorithms recently reported in the literatures, including GA, PSO and ACO. These algorithms are implemented in Matlab 7.1 according to the following parameters:

- Tissue-like P systems. Each cell contains 100 objects and communicates its first five best objects into two adjacent cells. The maximum computing step number is chosen to be 200. In the implementation, evolution rules uses the adaptive crossover probability p_c and mutation probability p_m. In order to study performances of tissue-like P systems of different degrees, four cases are considered in the experiments: $q = 4, 8, 16, 20$.

- GA [10]. The rotating wheel method, single-point crossover and single-point mutation are used, where the crossover and mutation probabilities, p_c and p_m, are chosen to be 0.8 and 0.001 respectively. Let the population size be $N_{swarm} = 100$ and maximum iteration number be $t_{max} = 200$.
- PSO [15]. The ω uses a linear decreasing inertia weight, where $\omega_{min} = 0.4$ and $\omega_{max} = 0.9$. $c_1 = c_2 = 2.0$, the population size $NP = 100$, and maximum iteration number is 200.
- ACO [16]. The best parameter values are $\gamma_1 = \gamma_2 = 1.0$ and $\rho = 0.99$.

4.3 Experimental results

Table 2. The performance comparisons of tissue-like P systems of different degrees.

Data sets	4 cells	8 cells	16 cells	20 cells
AD_5_2	327.01	326.94	326.44	326.94
	±0.0944	±0.0277	±0.0105	±0.0312
Data_9_2	591.11	591.12	591.06	591.03
	±0.1331	±0.0510	±0.0280	±0.0537
Square_4	2380.25	2380.26	2379.74	2380.00
	±0.1334	±0.0956	±0.0189	±0.0729
Sym_3_22	1248.31	1248.11	1247.72	1248.05
	±0.3156	±0.0554	±0.0105	±0.0333
Iris	96.84	96.81	96.75	96.77
	±0.0751	±0.0435	±0.0428	±0.0361
BreastCancer	2974.24	2971.14	2970.24	2969.06
	±1.5431	±1.5287	±1.1225	±1.0970
Newthyroid	1885.69	1870.37	1869.29	1871.18
	±14.3773	±1.7355	±0.9215	±2.2496
LungCancer	124.69	124.69	124.69	124.69
	±0.0045	±0.0012	±0.0011	±0.0035
Wine	16309.01	16303.42	16292.25	16301.97
	±2.5053	±1.9595	±0.1529	±2.8563
LiveDisorder	9860.54	9859.02	9851.78	9857.08
	±5.7239	±0.5116	±0.0347	±0.1043

In the experiments, we realize four tissue-like P systems with degrees 4, 8, 16 and 20 respectively. The aim is to evaluate the effects of the number of cells (i.e., different degrees) on clustering quality. The four tissue-like P systems are applied to find out the optimal cluster centers for the ten data sets respectively. In this work, the M value is also used to measure the clustering quality of each clustering algorithm. Considering that the evolution rules in the designed tissue-like P system include stochastic mechanism, we independently execute the tissue-like P systems of the four degrees 50 times on each data set, and then compute their mean values and standard deviations of the 50 runs. The mean values are used to illustrate the

average performance of the algorithms while standard deviations indicate their robustness. Table 2 provides experimental results of the tissue-like P systems of four degrees on ten data sets respectively. The results of degrees 16 and 20 are better than those of other two degrees, namely, lower mean values and smaller standard deviations. It can be further observed that the tissue-like P system with degree 16 obtains the smallest mean values and standard deviations on most of data sets. The results illustrate that the tissue-like P system with degree 16 has good clustering quality and high robustness.

Table 3. The results obtained by the algorithms for 50 runs on the ten data sets.

Data sets	P systems	GA	PSO	ACO	k-means
AD_5_2	326.44	332.31	326.44	326.45	332.47
	±0.0105	±0.4792	±0.0128	±0.0344	±3.1286
Data_9_2	591.06	593.7251	591.14	591.42	623.57
	±0.0280	±0.2635	±0.0303	±0.0372	±3.1326
Square_4	2379.74	2380.33	2379.74	2379.79	2386.00
	±0.0189	±0.6319	±0.0226	±0.0428	±4.5217
Sym_3_22	1247.72	1249.36	1247.72	1247.75	1255.45
	±0.0105	±1.2163	±0.0149	±0.0315	±3.8725
Iris	96.75	99.83	97.23	97.25	104.11
	±0.0428	±5.5239	±0.3513	±0.4152	±12.4563
BreastCancer	2970.24	3249.26	3050.04	3046.06	3251.21
	±1.1225	±229.734	±110.801	±90.500	±251.143
Newthyroid	1869.29	1875.11	1872.51	1872.56	1886.25
	±0.9215	±13.5834	±11.0923	±11.1045	±16.2189
LungCancer	124.69	129.52	127.23	127.31	139.40
	±0.0011	±4.4961	±1.1528	±1.2936	±7.3136
Wine	16292.25	16298.42	16292.25	16292.25	16312.43
	±0.1529	±2.1523	±0.1531	±0.1672	±9.4269
LiveDisorder	9851.73	9856.14	9851.73	9851.74	9868.32
	±0.0347	±1.9523	±0.0356	±0.0692	±7.9274

In order to further evaluate clustering performance, the proposed membrane clustering algorithm is compared with GA-based, PSO-based and ACO-based clustering algorithms as well as classical k-means algorithm. Tables 3 gives the comparison results of the tissue-like P system of degree 16 with other four clustering algorithms on the ten data sets, respectively. The comparison results show that the tissue-like P system provides the optimum average value and smallest standard deviation in compare to those of other algorithms. For instance, the results obtained on the *AD_5_2* show that the tissue-like P system converges to the optimum of 326.4478 at almost times and PSO reaches to 326.44 in most of runs, while ACO, GA and k-means attain 326.45, 322.31 and 332.47 respectively. The standard deviations of M values for the tissue-like P system, PSO and ACO are 0.0105, 0.0128 and 0.0344 respectively, which significantly are smaller than other two algorithms.

For the results on the *Iris*, the optimum value is 96.75, which is obtained in most of runs of the tissue-like P system, however, other four algorithms fail to attain the value even once within 50 runs. The results on the *Newthyroid* also show that the tissue-like P system provides the optimum value of 1869.29 while the PSO, ACO, GA and k-means obtain 1872.51, 1872.56, 1875.11 and 1886.25 respectively. In addition, the tissue-like P system obtains smallest standard deviation on each data set in compare to other four algorithms, which illustrates that it has high robustness.

The Wilcoxon's rank sum test is a nonparametric statistical significance test for independent samples. The statistical significance test has been conducted at the 5% significance level in the experiments. We create five groups for the ten data set, which are corresponding to the five clustering algorithms (tissue-like P system, GA, PSO, ACO and k-means) respectively. Each group consists of the M values produced by 50 consecutive runs of the corresponding algorithms. In order to illustrate the goodness is statistically significant, we have completed a statistical significance test for these clustering algorithms. Table 4 gives the p-values provided by Wilcoxon's rank sum test for comparison of two groups (one group corresponding to the tissue-like P system and another group corresponding to some other method) at a time. The null hypothesis assumes that there is no significant difference between the mean values of two groups, whereas there is significant difference in the mean values of two groups for the alternative hypothesis. It is evident from Table 4 that all p-values are less than 0.05 (5% significance level). This is a strong evidence against the null hypothesis, establishing significant superiority of the proposed membrane clustering algorithm.

Table 4. The results of p-values produced by Wilcoxon's rank sum test.

Data sets	GA	PSO	ACO	k-means
AD_5_2	4.1321×10^{-3}	2.3256×10^{-2}	2.6351×10^{-2}	3.4273×10^{-3}
Data_9_2	4.0536×10^{-3}	2.2734×10^{-2}	2.7932×10^{-2}	3.2963×10^{-3}
Square_4	3.9275×10^{-3}	2.1482×10^{-2}	2.8175×10^{-2}	3.5387×10^{-3}
Sym_3_22	3.7894×10^{-3}	2.4357×10^{-2}	2.8529×10^{-2}	3.4416×10^{-3}
Iris	4.0968×10^{-3}	3.5823×10^{-2}	3.2634×10^{-2}	3.6528×10^{-3}
BreastCancer	3.9235×10^{-3}	2.9527×10^{-2}	2.8192×10^{-2}	3.4632×10^{-3}
Newthyroid	3.8864×10^{-3}	2.5162×10^{-2}	2.9355×10^{-2}	3.5381×10^{-3}
LungCancer	3.8575×10^{-3}	2.7346×10^{-2}	2.7358×10^{-2}	3.5138×10^{-3}
Wine	3.7639×10^{-3}	3.2189×10^{-2}	2.7963×10^{-2}	3.6348×10^{-3}
LiveDisorder	3.8398×10^{-3}	2.4671×10^{-2}	2.8846×10^{-2}	3.5822×10^{-3}

5 Conclusion

In this paper, we discuss a membrane clustering algorithm, a novel clustering algorithm under the framework of membrane computing. Distinguished from the

existing evolutionary clustering techniques, two inherent mechanisms of membrane computing are exploited to realize the membrane clustering algorithm, including evolution and communication mechanisms. For this purpose, a tissue-like P system consisting of q cells is designed, in which each cell as parallel computing unit runs in maximally parallel way and each object of the system expresses a group of candidate cluster centers. Moreover, the communication rules impliedly realize a local neighborhood structure, namely, each cell exchanges and shares the best objects with its two adjacent cells. Under the control of evolution and communication mechanisms of objects, the tissue-like P system is able to search for the optimal cluster centers for a data set to be clustered. In addition, the local neighborhood structure can guide the exploitation of the optimal object and enhance the diversity of evolution objects. Therefore, the membrane clustering presented in this paper can be viewed as a successful instance for building a bridge between membrane computing and data clustering.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61170030), the Chunhui Project Foundation of the Education Department of China (Nos. Z2012025 and Z2012031), and the Sichuan Key Technology Research and Development Program (No. 2013GZX0155), China.

References

1. J.A. Hartigan, *Clustering Algorithm*, New York: Wiley, 1975.
2. A.K. Jain, R.C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Engiewood Cliffs, NJ, 1988.
3. R. Xu, D. Wunsch, Survey of clustering algorithm, IEEE Trans. Neural Networks 16(3) (2005) 645-678.
4. A.K. Jain, Data clustering: 50 years beyond k-means, Pattern Recognition Letters 31 (2010) 651-666.
5. B. Everitt, S. Landau, M. Leese, *Cluster Analysis*, Arnold, London, 2001.
6. S. Saha, S. Bandyopadhyay, A symmetry based multiobjective clustering technique for automatic evolution of clusters, Pattern Recognition 43 (2010) 738-751.
7. T. Kanungo, D. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, A. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Trans. Pattern Anal. Mach.Intell. 24(7) (2002) 881-892.
8. D. Steinley, K-meams clustering: A half-century synchesis, British Journal of Mathematical and Statistics Psychology 59(34) (2006) 1-34.
9. X. Wu, Top ten algorithms in data mining, Taylor & Francis, 2009.
10. S. Bandyopdhyay, U. Maulik, An evolutionary technique based on k-means algorithm for optimal clustering in RN, Inf. Sci. 146 (2002) 221-237.
11. S. Bandyopdhyay, S. Saha, GAPS: a clustering method using a new point symmetry-based distance measure, Pattern Recognition 40 (2007) 3430-3451.

12. M. Laszlo, S. Mukherjee, A genetic algorithm that exchanges neighboring centers for k-means clustering, Pattern Recognition Lett. 28 (2007) 2359-2366.
13. D. Chang, X. Zhang, C. Zheng, A genetic algorithm with gene rearrangment for k-means clustering, Pattern Recognition 42 (2009) 1210-1222.
14. C.D. Nguyen, K.J. Cios, GAKREM: A novel hybrid clustering algorithm, Information Sciences 178 (2008) 4205-4227.
15. Y.T. Kao, E. Zahara, I.W. Kao, A hybridized approach to data clustering, Expert Systems with Applications 34(3) (2008) 1754-1762.
16. P.S. Shelokar, V.K. Jayaraman, B.D. Kulkarni, An ant colony approach for clustering, Analytica Chimica Acta 509(2) (2004) 187-195.
17. T. Niknam, B. Amiri, An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis, Applied Soft Computing 10 (2010) 183-197.
18. Gh. Păun, Computing with membranes, Journal of Computer System Sciences 61(1) (2000) 108-143.
19. Gh. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook of Membrane Computing*, Oxford University Press, New York, 2010.
20. M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundameta Informaticae 71(2-3) (2006) 279-308.
21. R. Freund, Gh. Păun, M.J. Pérez-Jiménez, Tissue-like P systems with channel-states, Theoretical Computer Science 330(1) (2005) 101-116.
22. H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, Information Sciences 235 (2013) 106-116.
23. J. Wang, P. Shi, H. Peng, Mario J. Pérez-Jiménez, T. Wang, Weighted fuzzy spiking neural P systems, IEEE Transactions on Fuzzy Systems 21(2) (2013) 209-220.
24. Gh. Păun, M.J. Pérez-Jiménez, Membrane computing: Brief introduction, recent results and applications, BioSystems 85 (2006) 11-22.
25. L. Huang, I. Suh, A. Abraham, Dynamic mul-objective optimization based on membrane computing for control of time-varying unstable plants, Information Sciences, 181(11) (2011) 2370-2391.
26. G. Zhang, J. Cheng, M. Gheorghe, Q. Meng, A hybrid approach based on different evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, Applied Soft Computing, 13(3) (2013) 1528-1542.
27. C.A. Murthy, N. Chowdhury, In search of optimal clusters using genetic algorithms, Parttern Recognition Letters 17 (1996) 825-832.
28. U. Maulik, S. Bandyopadhyay, Genetic algorithm based clustering technique, Pattern Recognition 33 (2000) 1455-1465.
29. E. Falkenauer, Genetic Algorithms and Grouping Problems, John Wiley & Sons, 1998.
30. L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.
31. Z. Michalewicz, *Genetic Algorithm + Data Structure = Evolution Program*, Springer, New York, 1996.
32. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
33. R.A. Fisher, The use of multiple measurements in taxonomic problems, Ann. Eugenics 3 (1936) 179-188.
34. S.K. Pal, D.D. Majumder, Fuzzy sets and decision making approaches in vowel and speaker recognition, IEEE Trans. Systems, Man Cybernet. SMC-7 (1977) 625-629.
35. M. Clerc, J. Kennedy, The particle swarm explosion stability and convergence in a multi-dimensional complex space, IEEE Trans. Evolutionary Comput. 6(1) (2002) 58-73.

Application of Weighted Fuzzy Reasoning Spiking Neural P Systems to Fault Diagnosis in Traction Power Supply Systems of High-speed Railways

Tao Wang[1], Gexiang Zhang[1], Mario J. Pérez-Jiménez[2]

[1]School of Electrical Engineering, Southwest Jiaotong University,
Chengdu, 610031, P.R. China
email: wangtaocdu@gmail.com, zhgxdylan@126.com
[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Sevilla, 41012, Spain
email: marper@us.es

Summary. This paper discusses the application of weighted fuzzy reasoning spiking neural P systems (WFRSN P systems) to fault diagnosis in traction power supply systems (TPSSs) of China high-speed railways. Four types of neurons are considered in WFRSN P systems to make them suitable for expressing status information of protective relays and circuit breakers, and a weighted matrix-based reasoning algorithm (WMBRA) is introduced to fulfill the reasoning based on the status information to obtain fault confidence levels of faulty sections. Fault diagnosis production rules in TPSSs and their WFRSN P system models are proposed to show how to use WFRSN P systems to describe different kinds of fault information. Building processes of fault diagnosis models for sections and fault region identification of feeding sections, and parameter setting of the models are described in detail. Case studies including normal power supply and over zone feeding show the effectiveness of the presented method.

1 Introduction

Membrane computing, formally introduced by Gh. Păun in [1], is an attractive research field of computer science aiming at abstracting computing models, called membrane systems or P systems, from the structures and functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures. Spiking neural P systems (SN P systems), introduced in [2] in the framework of membrane computing, is a new class of computing devices which are inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons. Since then, SN P systems have become a hot topic in membrane computing [3]-[19], among which there are several investigations focus on the use of SN P systems and their variants to solve engineering problems.

In [14], a fuzzy reasoning spiking neural P system with real numbers was presented to fulfill diagnosis knowledge representation and reasoning. In [15], the FRSN P system was used for fault diagnosis in power systems and three different applications verified its effectiveness. Adaptive fuzzy spiking neural P systems (AFSN P systems) for fuzzy inference and learning were presented in [16] and the work in [17] focused on the application of AFSN P systems in fault diagnosis of power systems. The aforementioned investigations verify the feasibility and effectiveness of extended SN P systems in fault diagnosis of power systems.

Traction power supply systems (TPSSs) of high-speed railways are a kind of special power systems. In recent decades, the most studied intelligent fault diagnosis method for TPSSs of China high-speed railways is expert system (ES) [20]-[22]. ES expresses operation logic of protective relays and circuit breakers easily, and makes full use of experts knowledge, but it has a slow inference speed due to its sequential search nature, and the difficulties of designing and maintaining a rule-based knowledge system. In [23], fuzzy Petri nets (FPNs) are applied in the fault diagnosis of TPSSs to avoid the weakness of ESs and a second reasoning method was designed to improve the reliability of diagnosis results when status information of protections contains uncertainty and incompleteness. However, the second reasoning adds the complexity of computation in reasoning process and makes the fault diagnosis need more time. So, how to improve the aforementioned methods and explore new ones to solve fault diagnosis problems in TPSSs is worth further discussing.

This paper discusses the application of a novel and bioinspired model, weighted fuzzy reasoning spiking neural P systems (WFRSN P systems), to fault diagnosis in TPSSs. WFRSN P systems were first proposed in [18] and new ingredients, such as fuzzy truth value, weighted fuzzy logic, output weight, threshold and two types of neurons, were added to the original definition of SN P systems. Besides, a weighted fuzzy backward reasoning algorithm was developed for the WFRSN P systems to fulfil dynamic fuzzy reasoning. However, two types of neurons in WFRSN P systems can not express status information of protections completely. Moreover, the weighted fuzzy backward reasoning algorithm is too complex to use it to diagnose faults in TPSSs directly. To adapt WFRSN P systems to solve fault diagnosis problems in TPSSs, four types of neurons are considered in this study and a weighted matrix-based reasoning algorithm (WMBRA) is introduced to fulfill the reasoning of fault information. WFRSN P system models for fault diagnosis production rules in TPSSs are proposed to show how to describe different kinds of fault information by using them. How to build fault diagnosis models for sections and set parameters in the models are described in detail. Besides, owing to the special power supply manner of TPSSs, a WFRSN P system model for fault region identification of feeding sections is also built. Case studies show the effectiveness of the presented method.

The remainder of this paper is organized as follows. Section 2 states the problem to solve. The WFRSN P systems and WMBRA are defined in Section 3. Section 4 presents the key issues of fault diagnosis based on WFRSN P systems, and

the application of WFRSN P systems to fault diagnosis in TPSSs is discussed in Section 5. Conclusions are finally drawn in Section 6.

2 Problem Description

When faults occur in a power system, protective relays detect the faults and trip their corresponding circuit breakers (CBs) to isolate faulty sections from the operation of this power system and guarantee the other parts can operate normally. The aim of fault diagnosis in this paper is to identify the faulty sections by using status information of protective relays and CBs which are read from supervisor control and data acquisition (SCADA) systems. The framework of fault diagnosis in power systems using reasoning model-based method is shown as in Fig. 1 [15, 24]. There are three important parts in this framework: real-time data, static data and a flowchart of identification fault sections. The real-time data, protective relay operation information and circuit breaker tripping information, are used to estimate the outage areas to obtain candidate faulty sections using a network topology analysis method, so as to reduce the subsequent computational burden [24]. The static data, network topology and protection configuration of a power system, are used to build a fault diagnosis model for each candidate section in each outage area. The inputs of each diagnosis model are initialized by both real-time data and static data. Then, each diagnosis model performs reasoning algorithm to obtain fault confidence levels of candidate faulty sections to determine faulty sections. The diagnosis results include the faulty sections and their fault confidence levels.

TPSSs of high-speed railways are a kind of special power systems. Thus, the fault diagnosis of TPSSs of China high-speed railways can keep the framework of fault diagnosis in Fig. 1. Meanwhile, it is important to describe the characteristics and protection configuration of TPSSs because of their particularity.

The electrical principle schematic illustration for TPSSs of China high-speed railways is shown in Fig. 2. Power systems supply TPSSs with three-phase alternating currents (three-phase ACs) (220 kv or 330 kv) which are converted to single-phase alternating currents (single-phase ACs) (25 kv) by traction substations (TSSs) [25, 26]. Then, the single-phase ACs are supplied to electric locomotives by feeding sections. Subsection posts (SSPs) usually are built near pivotal station yards who need to output multipath feeder lines. The SSPs only has power distribution equipments because they are used only for the redistribution of power supply and do not convert the voltage. Thus, the function of SSPs is the same as power distribution stations. In order to increase the flexibility of power supply as well as to improve the reliability of electric power operation, section posts (SPs) usually are built between two TSSs. Over zone feeding is fulfilled by using over zone switches in SPs, and up and down line parallel power supply is fulfilled by using an up and down line interconnection switch, which can improve the reliability of TPSSs. Feeder lines connect TSSs and contact wires to transmit the electricity

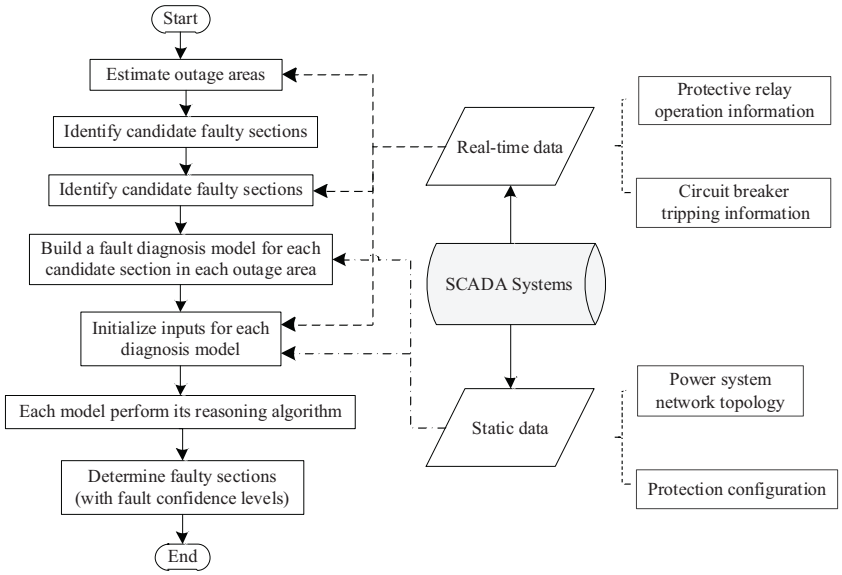


Fig. 1. Framework of fault diagnosis in power systems using reasoning model-based method.

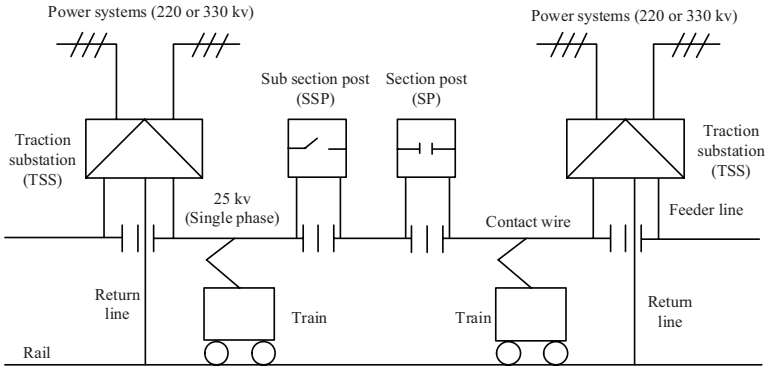


Fig. 2. Electrical principle schematic illustration of TPSSs.

from TSSs to contact wires. Return lines connect TSSs and rails to return loop currents in rails into TSSs. Traction networks are composed of feeder lines, contact wires, return lines, rails and earth ground.

If a TPSS uses autotransformer (AT) feeding manner, then one autotransformer is installed along rails every other 10 to 15 kilometers [25]. Fig. 3 shows a composition diagram of TPSSs in AT feeding manner considered in this study. Typical feeding sections in AT feeding manner consist of TSSs, ATPs and SPs. In this kind of TSS-ATP-SP feeding sections, lines in SPs are connected in an up and

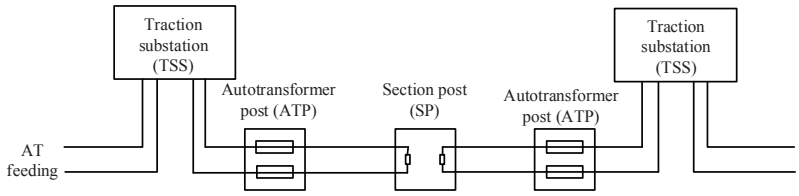


Fig. 3. Composition diagram of TPSSs in AT feeding manner.

Table 1. Protection configuration of feeder lines in AT TPSSs with normal power supply

Posts	Protection configurations		Automatic devices
TSSs	Main	One impedance protection Overcurrent protection of low voltage starting	One shot automatic reclosing
	Backup	$\triangle I$ current increment protection	
SPs	Main	No-voltage protection	Voltage checking for automatic reclosing
ATPs	Main	No-voltage protection	Voltage checking for automatic reclosing

down line paralleling manner while lines in ATPs are not. It is worth pointing out that if a feeding arm is very long, then SSPs are built between TSSs and SPs to reduce the power failure scope when faults occur in traction networks.

In this study, a feeding section fed by a same traction substation is considered as one fault diagnosis unit because the feeding section works in a relatively independent way in its operation, protection, etc. The faults on feeder lines, buses, traction transformers and autotransformers, in TSSs, ATPs and SPs, are diagnosed by using the status information read from SCADA systems, as shown in Fig. 1. Tables 1 and 2 show the protection configuration of feeder lines in AT TPSSs with normal power supply and over zone feeding, respectively [25]-[28]. Table 3 shows the protection configuration of transformers (traction transformers and autotransformers) in AT TPSSs [25]-[28].

When an AT traction power supply system is in the over zone feeding manner, the feeding section fed by the faulty TSS is temporarily fed by its adjacent TSS through over zone switches. When remote backup protections of feeder lines in TSSs operate, in order to distinguish which autotransformer (the one in the original TSS or in the temporary TSS) has a fault, main protections and primary backup protections of this autotransformer and their corresponding CBs are considered as reasoning conditions of the results obtained according to the remote backup protections.

Table 2. Protection configuration of feeder lines in AT TPSSs with over zone feeding

Posts	Protection configurations		Automatic devices
TSSs	Main	Two impedance protection Overcurrent protection of low voltage starting	One shot automatic reclosing
	Backup	$\triangle I$ current increment protection	
SPs	Main	One impedance protection Overcurrent protection of low voltage starting	Voltage checking for automatic reclosing
	Backup	$\triangle I$ current increment protection No-voltage protection	
ATPs	Main	No-voltage protection	Voltage checking for automatic reclosing

Table 3. Protection configuration of transformers in AT TPSSs

Transformers	Protection configurations	
Traction transformers	Main	Differential current quick-break protection Ratio-restrained differential protection of second harmonic lock Non-electrical protection
	Backup	Overcurrent protection of low voltage starting
Autotransformers	Main	Differential current quick-break protection Non-electrical protection
	Backup	Overcurrent protection of low voltage starting Shell collision protection

3 WFRSN P systems

3.1 Definitions

Definition 1: A WFRSN P system of $m \geq 1$ is a construct $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out)$, where:

(1) $O = \{a\}$ is a singleton alphabet (a is called spike);
(2) $\sigma_1, \ldots, \sigma_m$ are neurons, of the form $\sigma_i = (\theta_i, c_i, \overrightarrow{\omega_i}, \lambda_i, r_i), 1 \leq i \leq m$, where:
   a) θ_i is a real number in [0,1] representing the potential value of spikes (i.e. value of electrical impulses) contained in neuron σ_i;
   b) c_i is a real number in [0,1] representing the truth value associated with neuron σ_i;

c) $\overrightarrow{\omega_i} = (\omega_{i1}, \ldots, \omega_{iN_i})$ is a real number vector in (0,1] representing the output weight vector of neuron σ_i, where ω_{ij} $(1 \leq j \leq N_i)$ represents the weight on jth output arc (synapse) of neuron σ_i and N_i is a real number representing the number of synapses starting from neuron σ_i.
d) λ_i is a real number in [0,1) representing the firing threshold of neuron σ_i;
e) r_i represents a firing (spiking) rule contained in neuron σ_i with the form $E/a^\theta \to a^\beta$, where θ and β are real numbers in [0,1], $E = \{a^n, \theta \geq \lambda_i\}$ is the firing condition. The firing condition means that if and only if neuron σ_i receives at least n spikes and the potential value of spikes is with $\theta \geq \lambda_i$, then the firing rule contained in the neuron can be applied, otherwise, the firing rule cannot be applied;

(3) $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in syn, 1 \leq i, j \leq m$; that is, syn provides a (weighted) directed graph whose set of nodes is $\{1, \ldots, m\}$;
(4) $in, out \subseteq \{1, \ldots, m\}$ indicate the input neuron set and the output neuron set of Π, respectively.

How WFRSN P systems are extended from SN P systems are described as follows. First, the definition of neurons are extended. WFRSN P systems consist of two kinds of neurons, i.e., proposition neurons and rule neurons, where rule neurons contain three subcategories: *general*, *and* and *or*. Second, the pulse value θ_i contained in each neuron σ_i is a real numbers in [0,1] representing potential value of spikes contained in this neuron instead of the number of spikes in SN P systems. Third, each neuron is associated with either a proposition or a production rule, and $c_i \in [0, 1]$ represents the truth value of this proposition or the certainty factor (CF) of this production rule. Fourth, each weighted directed synapse has an output weight. In other words, each synapse in $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$ has a weight. The output weights of neurons represent the importance degree of their values in contributing to the computing results in output neurons. Fifth, each neuron contains only one firing (spiking) rule of the form $E/a^\theta \to a^\beta$. When the firing condition of one neuron is satisfied, the firing rule is applied, which means that the potential value θ is consumed and then this neuron produces a new spike with potential value of β. These different types of neurons aforementioned handle the potential values θ and β in different ways (see definition *Definition 2-5*). If the firing condition of one neuron is not satisfied, then the potential value of the spikes received by this neuron is updated via logical *and* or *or* operators. Finally, time delay is ignored in WFRSN P systems, thus all neurons are always open.

The definitions of different types neurons in WFRSN P systems are described as follows.

Definition 2: A *proposition neuron* is associated with a proposition in a fuzzy production rule. Such a neuron is represented by a circle and symbol P, as shown is Fig. 4.

If a *proposition neuron* is an input neuron of a WFRSN P system Π, then its potential value θ is received from the environment; otherwise, θ equals to the result of logical *or* operation on all weighted potential values received from its presynaptic rule neurons, i.e., $\theta = max\{\theta_1 * \omega_1, \ldots, \theta_k * \omega_k\}$. The firing rule of

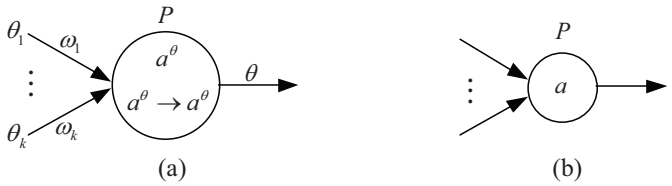


Fig. 4. A *proposition neuron* (a) and its simplified form (b).

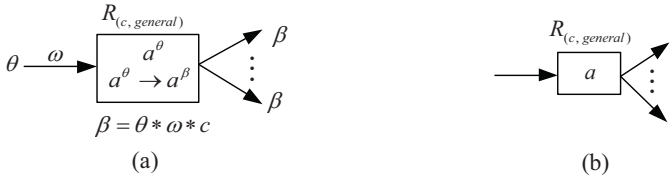


Fig. 5. A *general rule neuron* (a) and its simplified form (b).

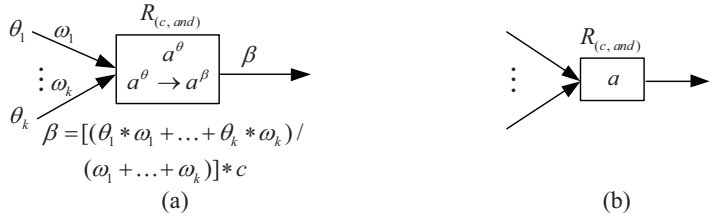


Fig. 6. An *and rule neuron* (a) and its simplified form (b).

a *proposition neuron* is of the form $E/a^\theta \to a^\theta$, in other words, the parameter β of the firing rule contained in such a neuron is identical to θ. When the firing condition E of a *proposition neuron* is satisfied, the potential value θ of spikes contained in this neuron is consumed and then a new spike with potential value θ is produced and emitted.

Definition 3: A *general rule neuron* is associated with a fuzzy production rule which has only one proposition in the antecedent part of the rule. Such a neuron is represented by a rectangle and symbol $R_{(c,\ general)}$), as shown is Fig. 5.

A *general rule neuron* has only one presynaptic proposition neuron and one or more postsynaptic proposition neurons. If a *general rule neuron* receives a spike from its presynaptic proposition neuron and its firing condition is satisfied, then the neuron fires and produces a new spike with the potential value $\beta = \theta * \omega * c$.

Definition 4: An *and rule neuron* is associated with a fuzzy production rule which has more than one propositions with an *and* relationship in the antecedent part of the rule. Such a neuron is represented by a rectangle and symbol $R_{(c,\ and)}$, as shown is Fig. 6.

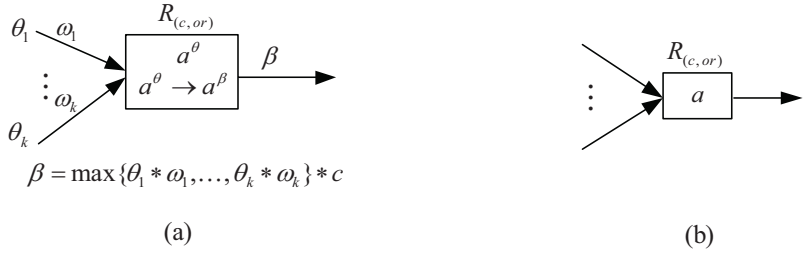


Fig. 7. An *or rule neuron* (a) and its simplified form (b).

An *and rule neuron* has more than one presynaptic proposition neurons and only one postsynaptic proposition neuron. If an *and rule neuron* receives k spikes from its k presynaptic proposition neurons and its firing condition is satisfied, then the neuron fires and produces a new spike with the potential value $\beta = [(\theta_1 * \omega_1 + \ldots + \theta_k * \omega_k)/(\omega_1 + \ldots + \omega_k)] * c$.

Definition 5: An *or rule neuron* is associated with a fuzzy production rule which has more than one propositions with an *or* relationship in the antecedent part of the rule. Such a neuron is represented by a rectangle and symbol $R_{(c,\ or)}$, as shown is Fig. 7.

An *or rule neuron* has more than one presynaptic proposition neurons and only one postsynaptic proposition neuron. If an *or rule neuron* receives k spikes from its k presynaptic proposition neurons and its firing condition is satisfied, then the neuron fires and produces a new spike with the potential value $\beta = max\{\theta_1 * \omega_1, \ldots, \theta_k * \omega_k\} * c$.

3.2 WMBRA

In order to clearly present a weighted matrix-based reasoning algorithm (WMBRA), we first introduce some parameter vectors and matrices as follows.

(1) $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_s)^T$ is a real truth value vector of the s proposition neurons, where θ_i $(1 \leq i \leq s)$ is a real number in [0, 1] representing the potential value contained in the ith proposition neuron. If there is not any spike contained in a proposition neuron, its potential value is 0.

(2) $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_t)^T$ is a real truth value vector of the t rule neurons, where δ_j $(1 \leq j \leq t)$ is a real number [0, 1] representing the potential value contained in the jth rule neuron. If there is not any spike contained in a rule neuron, its potential value is 0.

(3) $\boldsymbol{C} = diag(c_1, c_2, \ldots, c_t)$ is a diagonal matrix, where c_j $(1 \leq j \leq t)$ is a real number in [0,1] representing the certainty factor of the jth fuzzy production rule,

(4) $\boldsymbol{W}_{r1} = (\omega_{ij})_{s \times t}$ is a synaptic weight matrix representing the directed connection with weights among *proposition neurons* and *general rule neurons*. If there is a directed arc (synapse) from *proposition neuron* σ_i to *general rule neuron* σ_j, then ω_{ij} is identical to the output weight of synapse (i, j), otherwise, $\omega_{ij} = 0$.

(5) $\boldsymbol{W}_{r2} = (\omega_{ij})_{s\times t}$ is a synaptic weight matrix representing the directed connection with weights among *proposition neurons* and *and rule neurons*. If there is a directed arc (synapse) from *proposition neuron* σ_i to *and rule neuron* σ_j, then ω_{ij} is identical to the output weight of synapse (i, j), otherwise, $\omega_{ij} = 0$.

(6) $\boldsymbol{W}_{r3} = (\omega_{ij})_{s\times t}$ is a synaptic weight matrix representing the directed connection with weights among *proposition neurons* and *or rule neurons*. If there is a directed arc (synapse) from *proposition neuron* σ_i to *or rule neuron* σ_j, then ω_{ij} is identical to the output weight of synapse (i, j), otherwise, $\omega_{ij} = 0$.

(7) $\boldsymbol{W}_p = (\omega_{ji})_{t\times s}$ is a synaptic weight matrix representing the directed connection with weights among rule neurons and proposition neurons. If there is a directed arc (synapse) from rule neuron σ_j to proposition neuron σ_i, then ω_{ji} is identical to the output weight of synapse (j, i), otherwise, $\omega_{ji} = 0$.

(8) $\boldsymbol{\lambda_p} = (\lambda_{p1}, \ldots, \lambda_{ps})^T$ is a threshold vector of the s proposition neurons, where λ_{pi} $(1 \leq i \leq s)$ is a real number in [0, 1) representing the firing threshold of the ith proposition neuron.

(9) $\boldsymbol{\lambda_r} = (\lambda_{r1}, \ldots, \lambda_{rt})^T$ is a threshold vector of the t rule neurons, where λ_{rj} $(1 \leq j \leq t)$ is a real number in [0, 1) representing the firing threshold of the jth rule neuron.

Subsequently, we introduce some multiplication operations as follows.

(1) $\otimes$: $\boldsymbol{W}_{rl}^T \otimes \boldsymbol{\theta} = (\bar{\omega_1}, \bar{\omega_2}, \ldots, \bar{\omega_t})^T$, where $\bar{\omega_j} = \omega_{1j} * \theta_1 + \ldots + \omega_{sj} * \theta_s$, $j = 1, \ldots, t$, $1 \leq l \leq 3$.

(2) $\oplus$: $\boldsymbol{W}_{rl}^T \oplus \boldsymbol{\theta} = (\bar{\omega_1}, \bar{\omega_2}, \ldots, \bar{\omega_t})^T$, where $\bar{\omega_j} = (\omega_{1j} * \theta_1 + \ldots + \omega_{sj} * \theta_s)/(\omega_{1j} + \ldots + \omega_{sj})$, $j = 1, \ldots, t$, $1 \leq l \leq 3$.

(3) $\odot$: $\boldsymbol{W}_{rl}^T \odot \boldsymbol{\theta} = (\bar{\omega_1}, \bar{\omega_2}, \ldots, \bar{\omega_t})^T$, where $\bar{\omega_j} = max\{\omega_{1j} * \theta_1, \ldots, \omega_{sj} * \theta_s)\}$, $j = 1, \ldots, t$, $1 \leq l \leq 3$. Likewise, $\boldsymbol{W}_p^T \odot \boldsymbol{\delta} = (\bar{\omega_1}, \bar{\omega_2}, \ldots, \bar{\omega_s})^T$, where $\bar{\omega_i} = max\{\omega_{1i} * \delta_1, \ldots, \omega_{ti} * \delta_t\}$, $i = 1, \ldots, s$.

Next, we list the pseudocode of WMBRA.

4 Fault Diagnosis Based on WFRSN P systems

The diagnosis strategy based on WFRSN P systems is to build a WFRSN P system fault diagnosis model for each candidate faulty section in outage areas. Each model performs WMBRA from a set of SCADA data to the diagnosis results in the form of faulty sections and their fault confidence levels. If the confidence level of a candidate faulty section is larger than 0.5, then this section is a faulty section. This section presents the key issues of fault diagnosis based on WFRSN P systems. We first present WFRSN P system models for fault diagnosis production rules in TPSSs. Subsequently, how to build WFRSN P system fault diagnosis models for sections in TPSSs and set parameters in the models are described in detail. Finally, a WFRSN P system model for fault region identification of feeding sections is built.

Algorithm WMBFRA

Input: $\boldsymbol{W}_{r1}$, $\boldsymbol{W}_{r2}, \boldsymbol{W}_{r3}$, $\boldsymbol{W}_p$, $\boldsymbol{\lambda}_p$, $\boldsymbol{\lambda}_r$, $\mathbf{C}$, $\boldsymbol{\theta}_0$, $\boldsymbol{\delta}_0$

Set the termination condition $\boldsymbol{0} = (0, \ldots, 0)_t^T$
Let $g = 0$, where g represents the reasoning step
while $\boldsymbol{\delta}_g \neq \boldsymbol{0}$ **do**
  for each input neuron ($g = 0$) or each proposition neuron ($g > 0$) **do**
    if the firing condition $E = \{a^n, \theta_i \geq \lambda_{pi}, 1 \leq i \leq s\}$ is satisfied **then**
      the neuron fires and compute the real truth value vector $\boldsymbol{\delta}_{g+1}$ via $\boldsymbol{\delta}_{g+1} = (\mathbf{W}_{r1}^T \otimes \boldsymbol{\theta}_g) + (\mathbf{W}_{r2}^T \oplus \boldsymbol{\theta}_g) + (\mathbf{W}_{r3}^T \odot \boldsymbol{\theta}_g)$
      if there is a postsynaptic rule neuron **then**
        the neuron transmits a spike to the next rule neuron
      else
        just accumulate the value in the neuron
      end if
    end if
  end for
  for each rule neuron **do**
    if the firing condition $E = \{a^n, \delta_j \geq \lambda_{rj}, 1 \leq j \leq t\ \}$ is satisfied **then**
      the rule neuron fires and computes the real truth value vector $\boldsymbol{\theta}_{g+1}$ via $\boldsymbol{\theta_{g+1}} = \mathbf{W}_p^T \odot (\mathbf{C} \otimes \boldsymbol{\delta_{g+1}})$ and transmits a spike to the next proposition neuron
    end if
    $g = g + 1$
  end for
end while

Output: $\boldsymbol{\theta}_g$, which represents the final states of pulse values contained in proposition neurons.

4.1 WFRSN P system models for fault diagnosis production rules in TPSSs

In what follows, we describe fault diagnosis production rules in TPSSs and their WFRSN P system models, as shown is Fig. 8.

Type 1 (*Simple Rules*) R_i: IF $p_j(\theta_j)$ THEN $p_k(\theta_k)$ (CF = c_i), where p_j and p_k are propositions, c_i is a real number in [0,1] representing the certainty factor of rule R_i, θ_j and θ_k are real numbers in [0,1] representing the truth values of p_j and p_k, respectively. The weight of proposition p_j is ω_j, where $\omega_j = 1$ because there is only one proposition in the antecedent of this kind of rules. The truth values of p_k is $\theta_k = \theta_j * \omega_j * c_i = \theta_j * c_i$.

Type 2 (*Compound And Rules*) R_i: IF $p_1(\theta_1)$ and ... and $p_{k-1}(\theta_{k-1})$ THEN p_k (θ_k) (CF = c_i), where $p_1, \ldots, p_k$ are propositions, c_i is a real number in [0,1] representing the certainty factor of rule R_i, $\theta_1, \ldots, \theta_k$ are real numbers in [0,1] representing the truth values of $p_1, \ldots, p_k$, respectively. The weights of propositions $p_1, \ldots, p_{k-1}$ are $\omega_1, \ldots, \omega_{k-1}$, respectively. The truth values of p_k is $\theta_k = [(\theta_1 * \omega_1 + \ldots + \theta_{k-1} * \omega_{k-1})/(\omega_1 + \ldots + \omega_{k-1})] * c_i$.

Type 3 (*Compound Or Rules*) R_i: IF $p_1(\theta_1)$ or ... or $p_{k-1}(\theta_{k-1})$ THEN $p_k(\theta_k)$ (CF = c_i), where $p_1, \ldots, p_k$ are propositions, c_i is a real number in [0,1] rep-

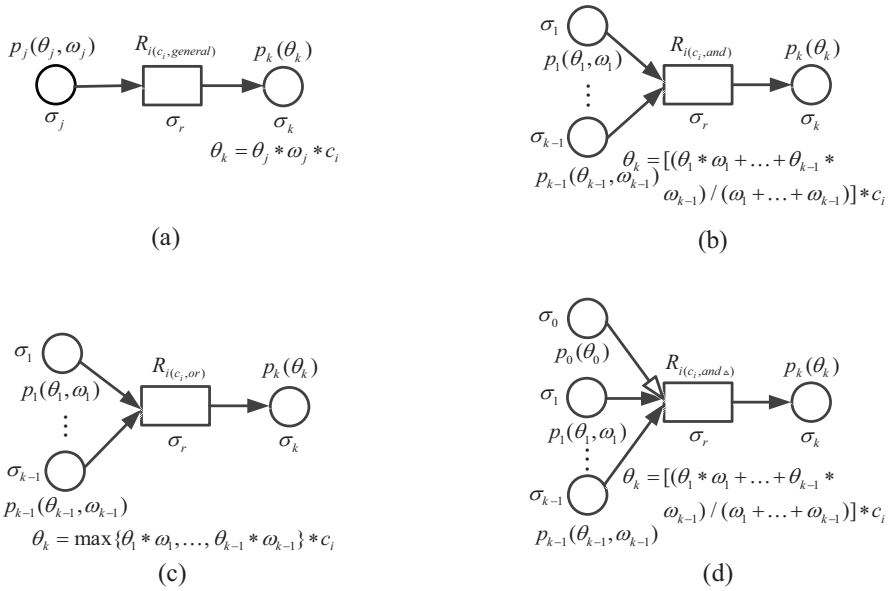


Fig. 8. WFRSN P system models for fault diagnosis production rules in TPSSs. (a) *Type 1*; (b) *Type 2*; (c) *Type 3*; (d) *Type4*.

resenting the certainty factor of rule R_i, $\theta_1, \ldots, \theta_k$ are real numbers in [0,1] representing the truth values of $p_1, \ldots, p_k$, respectively. The weights of propositions $p_1, \ldots, p_{k-1}$ are $\omega_1, \ldots, \omega_{k-1}$, respectively. The truth values of p_k is $\theta_k = max\{\theta_1 * \omega_1, \ldots, \theta_{k-1} * \omega_{k-1}\} * c_i$.

Type 4 (*Conditional And Rules*) R_i: WHEN $p_0(\theta_0)$ is true, IF $p_1(\theta_1)$ and ... and $p_{k-1}(\theta_{k-1})$ THEN $p_k(\theta_k)$ (CF = c_i), where $p_0, \ldots, p_k$ are propositions, c_i is a real number in [0,1] representing the certainty factor of rule R_i, $\theta_0, \ldots, \theta_k$ are real numbers in [0,1] representing the truth values of $p_0, \ldots, p_k$, respectively. The proposition p_0 is used to judge whether the reasoning condition of rule R_i is satisfied and its truth value θ_0 is not used in reasoning process. Thus, the weight of θ_0 is not considered in the model. The weights of propositions $p_1, \ldots, p_{k-1}$ are $\omega_1, \ldots, \omega_{k-1}$, respectively. The truth values of p_k is $\theta_k = [(\theta_1 * \omega_1 + \ldots + \theta_{k-1} * \omega_{k-1})/(\omega_1 + \ldots + \omega_{k-1})] * c_i$.

4.2 WFRSN P system fault diagnosis models for sections

A good WFRSN P system fault diagnosis model should be able to intuitively describe the causality between a fault and the the statues of its protective devices. Moreover, all kinds of protective devices including main protective relays, backup protective relays and their corresponding CBs of a faulty section should be considered in its diagnosis model. In order to show how to build models and set parameters, we take bus A in a TSS and its WFRSN P system fault diagnosis

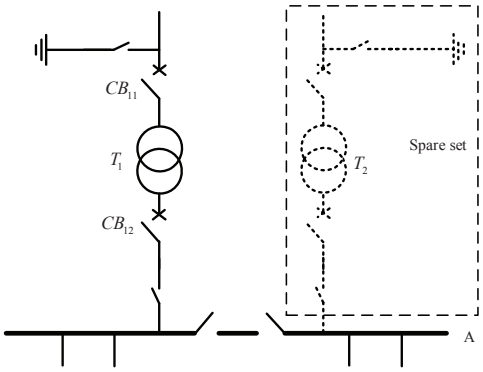


Fig. 9. Single line diagram of bus A in a TSS.

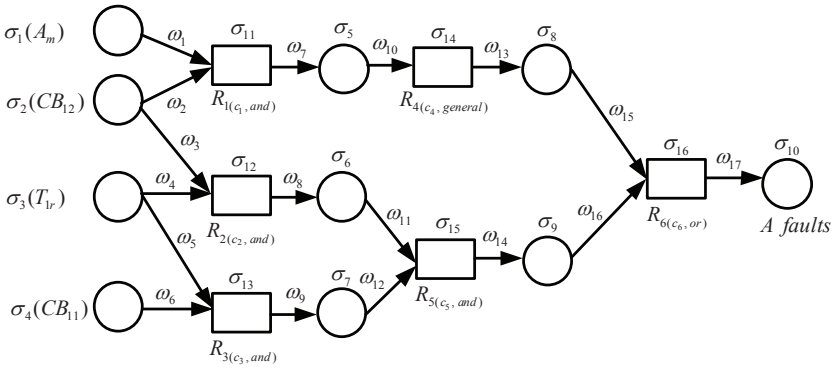


Fig. 10. A WFRSN P system fault diagnosis model for bus A.

model as examples which are shown in Fig. 9 and Fig. 10, respectively, where T represents a transformer, dotted line part represents a spare section set, m, p, r represent main protection, primary backup protection and remote backup protection, respectively.

Models building

When a fault occurs on a section in a TPSS, protective devices of this section will reach certain statues accordingly to protect the system. The observed status information, protective relay operation information and circuit breaker tripping information, obtained from SCADA systems are used as inputs of the WFRSN P system fault diagnosis model of the section. For example, in Fig. 10, main protective relay A_m, remote backup protective relay T_{1r} and their corresponding CBs, CB_{11} and CB_{12}, are used as the inputs of the diagnosis model of bus A. The other parts of the model are built according to relationships between the protective devices and the fault occurrence on bus A. For example, the relationships about

Table 4. Operation and non-operation confidence levels of the protective devices

Sections	Protective devices (operated)						Protective devices (non-operated)					
	Main		Primary backup		Remote backup		Main		Primary backup		Remote backup	
	Relays	CBs	Relays	CBs	Relays	CBs	Relays	CBs	Relays	CBs	Relays	CBs
FL	0.9913	0.9833	0.8	0.85	0.7	0.75	0.2	0.2	0.2	0.2	0.2	0.2
B	0.8564	0.9833	-	-	0.7	0.75	0.4	0.2	-	-	0.4	0.2
T	0.7756	0.9833	0.75	0.8	0.7	0.75	0.4	0.2	0.4	0.2	0.4	0.2

bus A can be described as follows: IF A_m operates and CB_{12} trips THEN bus A fails; IF T_{1r} operates and (CB_{11}, CB_{12} trip) THEN bus A fails. Then proposition neurons and different types of rule neurons are chosen, and their links are created according to the relationships to obtain the WFRSN P system fault diagnosis model in Fig. 10. Output neuron σ_{10} will export the fault confidence level of bus A once WMBRA stops.

A WFRSN P system for the model in Fig. 10 can be formally described as $\Pi_1 = (O, \sigma_1, \ldots, \sigma_{16}, syn, in, out)$, where:

(1) $O = \{a\}$ is the singleton alphabet (a is called spike).

(2) $\sigma_1, \ldots, \sigma_{10}$ are *proposition neurons* corresponding to the propositions with truth values $\theta_1, \ldots, \theta_{10}$; that is, $s = 10$.

(3) $\sigma_{11}, \ldots, \sigma_{16}$ are rule neurons, where σ_{11}, σ_{12}, σ_{13} and σ_{15} are *and rule neurons*, σ_{14} is a *general rule neuron* and σ_{16} is an *or rule neuron*; that is, $t = 6$.

(4) $syn = \{(1, 11), (2, 11), (2, 12), (3, 12), (3, 13), (4, 13), (5, 14), (6, 15), (7, 15), (8, 16), (9, 16), (11, 5), (12, 6), (13, 7), (14, 8),(15, 9),(16, 10)\}$.

(5) $in = \{\sigma_1, \ldots, \sigma_4\}$, $out = \{\sigma_{10}\}$.

Parameters setting

Since the protections of sections in TPSSs are designed in single-ended manner, the status information of protective devices obtained form SCADA systems may contain uncertainty and incompleteness caused by abnormal situations such as operation failure, malversation and misinformation. Thus, it is necessary to use a probability value to describe the operation confidence level of each section. In consideration of the generality of the reliability of protective relays and CBs in TPSSs and ordinary power systems, the operation confidence levels of these protective devices are set the same as those in [17, 23, 29]. Table 4 shows the confidence levels of operated protective devices and non-operate protective devices, where FL, B and T represent the feeder line, bus and transformer, respectively.

Initially, each input neuron of a WFRSN P system fault diagnosis model contains only one spike assigned a real number, which is identical with the confidence level of the protective device associated with this input neuron. The other neurons in the model do not contain spikes at the very beginning and their pulse values are 0. For example, in Fig. 10, if bus A_m and CB_{12} operate, and T_{1r}, CB_{11} do not operate, then the spikes contained in $\sigma_1, \ldots, \sigma_4$ are given the values of 0.8564,

0.9833, 0.4, 0.2, respectively. The pulse values of $\sigma_5, \ldots, \sigma_{16}$ are given the same value 0.

Each rule neuron of a WFRSN P system fault diagnosis model has a truth value which represents the certainty factor of the fault diagnosis production rule associated with this rule neuron. Usually, a main protection has a higher reliability than that of a primary backup protection while a primary backup protection has a higher reliability than that of a remote backup protection. The truth values of neurons associated with main, primary backup and remote backup protections are set as 0.975, 0.95, 0.9, respectively. It is worth pointing out that for *or rule neurons*, their truth values are set according to their highest protection. In Fig. 10, the truth values of $\sigma_{11}, \ldots, \sigma_{16}$ are set as 0.975, 0.9, 0.9, 0.975, 0.9, 0.975.

Since the protective relay operation information and circuit breaker tripping information are both important to a fault diagnosis production rule, the output weights of proposition neurons associated with protective relays and CBs are set as the same value 0.5. If a neuron has only one presynaptic neuron, then the output weight of its presynaptic neuron is set as 1. Besides, the weight of a protection type is also set as 1. The weighs $\omega_1, \ldots, \omega_{17}$ in Fig. 10 are set as 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1, 1, 1, 1, 0.5, 0.5, 1, 1, 1, 1, 1.

The firing threshold value of each neuron in WFRSN P system fault diagnosis models should be smaller than the minimum pulse value appeared in the neurons in the whole reasoning process. According to Table 4 and the operation of pulse values in different types of neurons, the firing threshold value of each neuron is set as 0.1.

4.3 Fault region identification for feeding sections

Lines in section posts (SPs) are connected in an up and down line paralleling manner in a TSS-ATP-SP feeding section (FS). So, when faults confirmed occur in a feeding section, one important task of fault diagnosis for taction power supply systems is to identify fault regions (which parts fail) in FSs. Fig. 11 shows a single line diagram of a TSS-ATP-SP feeding section and its WFRSN P system fault diagnosis model for fault region identification is shown in Fig. 12, where neurons σ_1 and σ_2 are associated with the propositions that current directions of I_{34} and I_{35} are positive, respectively; neurons σ_3 is associated with the proposition that current is detected in SP2; neurons σ_4 and σ_5 are associated with the propositions that current directions of I_{42} and I_{43} are negative, respectively; a small circle on an arrow tip represents an inverse proposition associated with its presynaptic neuron; a hollow tip represents an assistant synapse, i.e., the proposition associated with its presynaptic neuron is used as a judgement condition; output neuron σ_6 is associated with the proposition that first part of up direction feeding section in FS2, i.e., $FS2_{1\ up}$ has a fault. The meanings of output neurons σ_7, σ_9, σ_9 are similar. Here, clockwise direction is the positive current direction while counter-clockwise direction is the negative one.

Fig. 11 and Fig. 12 show a typical feeding section and its WFRSN P system fault diagnosis model for fault region identification, the models for other feeding

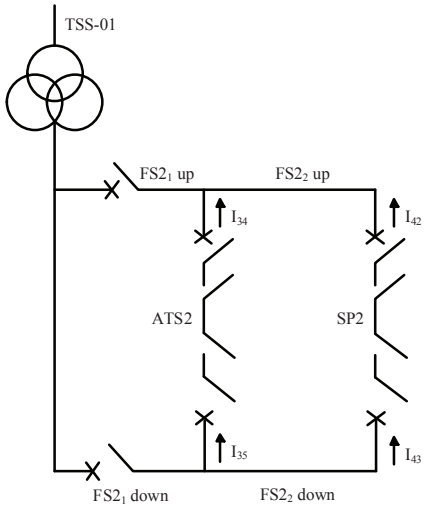


Fig. 11. Single line diagram of a TSS-ATP-SP feeding section.

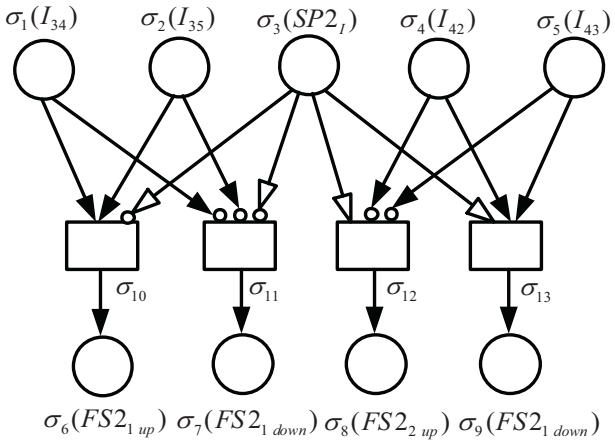


Fig. 12. A WFRSN P system fault diagnosis model for fault region identification of a feeding section.

sections can be built in a similar way. Causality between currents detected and fault regions is described by a WFRSN P system fault diagnosis model to get the fault regions of feeding sections and no numerical calculation is involved in this identification process. Thus, parameter setting of WFRSN P system fault diagnosis models for fault region identification of feeding sections is not considered.

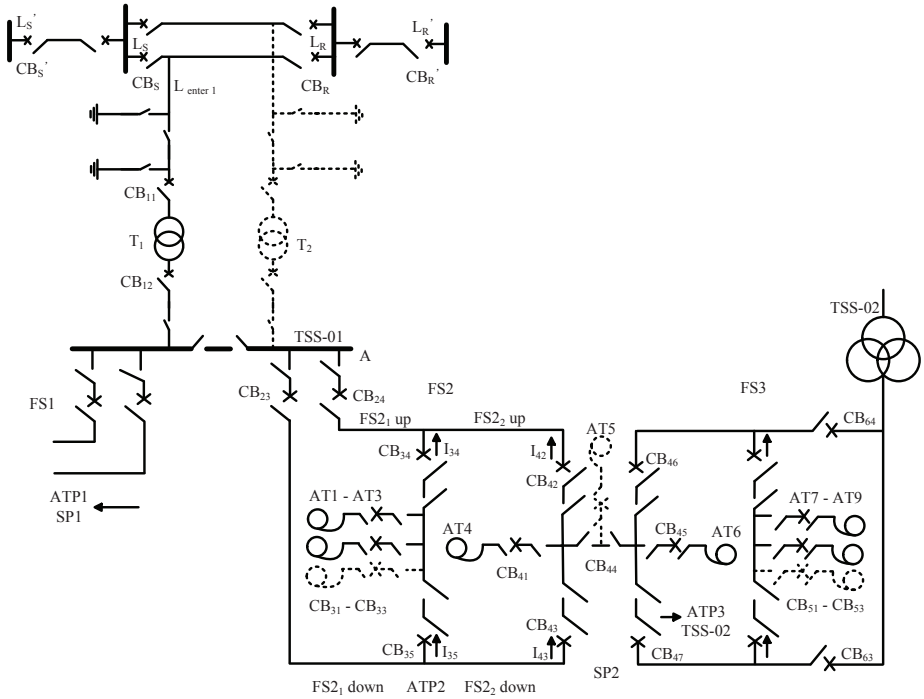


Fig. 13. A local single line sketch map of a TPSS.

5 Applications

In this section, three cases from the local system of a TPSS chosen in [23], as shown in Fig. 13, are considered as examples to test the effectiveness of WFRSN P systems in fault diagnosis, where S and R represent the sending end and receiving end of transmission lines, L represents transmission lines. The first two cases are in normal power supply and the third case is in over zone feeding. It is worth pointing out that, the complete line connection of FS1, ATP1, SP1, FS3, ATP3 and TPS-02 is the same as that of TSS-01, FS2, SP2 and ATP2 in Fig. 13.

Case 1: normal power supply. $FS2_{1\ up}$ and AT1 have faults.

Status information from the SCADA system (in time order): $AT1_m$ operated, CB_{31} tripped, $AT3$ auto switched over; $FS2_m$ operated, CB_{23} and CB_{24} tripped; feeder lines auto reclosed, $FS2_{up\ m}$ operated quickly, CB_{23} tripped again. When faults occur, current directions of I_{34} and I_{35} are positive, and current is not detected in SP2.

A WFRSN P system for $FS2_{up}$ is Π_2 and its corresponding WFRSN P system fault diagnosis model is shown in Fig. 14. $\Pi_2 = (O, \sigma_1, \ldots, \sigma_{16}, syn, in, out)$, where:

(1) $O = \{a\}$ is the singleton alphabet (a is called spike).

(2) $\sigma_1, \ldots, \sigma_9$ are proposition neurons corresponding to the propositions with truth values $\theta_1, \ldots, \theta_9$; that is, $s = 9$.

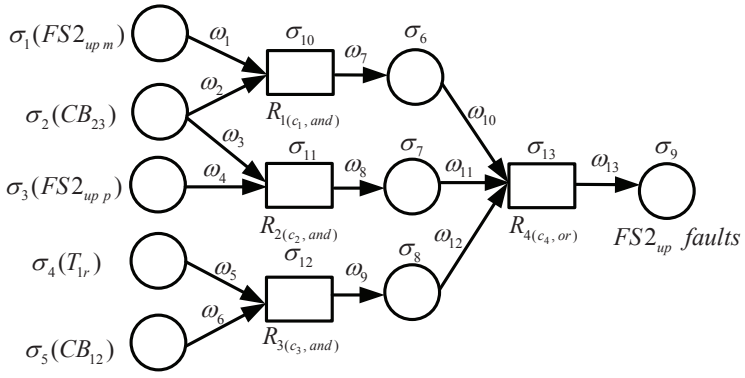


Fig. 14. A WFRSN P system fault diagnosis model for $FS2_{up}$.

(3) $\sigma_{10}, \ldots, \sigma_{13}$ are rule neurons, where σ_{10}, σ_{11} and σ_{12} are *and* rule neurons, σ_{14} is an *or* rule neuron; that is, $t = 4$.

(4) $syn = \{(1, 10)\ , (2, 10), (2, 11), (3, 11), (4, 12), (5, 12), (6, 13), (7, 13), (8, 13), (10, 6), (11, 7), (12, 8), (13, 9)\}$.

(5) $in = \{\sigma_1, \ldots, \sigma_5\}$, $out = \{\sigma_9\}$.

The synaptic weight matrices of Π_2 are shown in Fig. 15 and other parameter matrices associated with the model in Fig. 14 are described as follows: $\boldsymbol{\theta_0} = (0.9913\ 0.9833\ 0.8\ 0.4\ 0.2\ 0\ 0\ 0\ 0)^T$, $\boldsymbol{\delta_0} = (0\ 0\ 0\ 0)^T$, $\boldsymbol{C} = diag(0.975\ 0.95\ 0.9\ 0.975)$. In order to succinctly describe the matrices, let us denote $\boldsymbol{O_l} = (x_1, \ldots, x_l)^T$, where $x_i = 0, 1 \leq i \leq l$. When $g = 0$, we get the results: $\boldsymbol{\delta_1} = (0.9873\ 0.8917\ 0.3\ 0)^T$, $\boldsymbol{\theta_1} = (0\ 0\ 0\ 0\ 0\ 0.9626\ 0.8471\ 0.27\ 0)^T$. When $g = 1$, we get the results: $\boldsymbol{\delta_2} = (0\ 0\ 0\ 0.9626)^T$, $\boldsymbol{\theta_2} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0.9385)^T$. When $g = 2$, we get the results: $\boldsymbol{\delta_3} = (0\ 0\ 0\ 0)^T$. Thus, the termination condition is satisfied and the reasoning process ends. We obtain the reasoning results, i.e., the truth value 0.9385 of the output neuron σ_9. The feeding section $FS2_{up}$ has a fault with a fault confidence level 0.9385. The fault region of $FS2_{up}$ can be further identified according to the fault current detected and the WFRSN P system fault diagnosis model for fault region identification in Fig. 12, and then we get the result that $FS2_{1\ up}$ has a fault with a fault confidence level 0.9385.

For AT1, a WFRSN P system can be constructed in a similar way and its corresponding WFRSN P system fault diagnosis model is shown in Fig. 16. The diagnosis process of AT1 is similar. According to the SCADA data and Table 4, the parameter matrices of WFRSN P system fault diagnosis model for AT1 is established to perform WMBRA. After the reasoning, the fault confidence level of AT1 is obtained, i.e., 0.8361. So the autotransformer AT1 has a fault with a fault confidence level 0.8361.

Case 2: normal power supply. $FS2_{1\ up}$ has faults.

$$\boldsymbol{W_{r1}} = [\mathbf{O}]_{9\times 4}, \boldsymbol{W_{r2}} = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{W_{r3}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{W_p} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Fig. 15. Synaptic weight matrices of WFRSN P system fault diagnosis model for $FS2_{up}$.

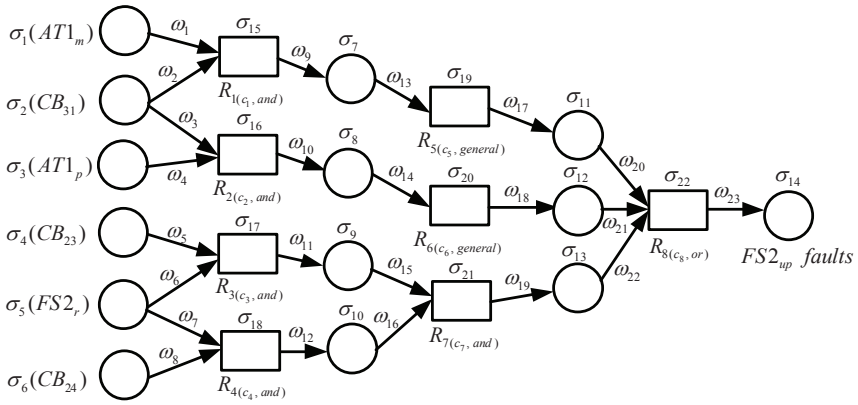


Fig. 16. A WFRSN P system fault diagnosis model for AT1.

Status information from the SCADA system (in time order): $FS2_m$ operated, CB_{24} tripped; T_{1r} operated, CB_{11} and CB_{12} tripped. When faults occur, current directions of I_{34} and I_{35} are positive, and current is not detected in SP2. In this case, CB_{23} refused operation.

According to the SCADA data and Table 4, the WFRSN P system fault diagnosis model for $FS2_1$ and its parameter matrices are established to perform WMBRA. After the reasoning, the fault confidence level of $FS2_{up}$ is obtained, i.e., 0.7439. The fault region of $FS2_{up}$ can be further identified according to the fault current detected and the WFRSN P system fault diagnosis model for fault region identification in Fig. 12, and then we get the result that $FS2_{1\ up}$ has a fault. So the feeding section $FS2_{1\ up}$ has a fault with a fault confidence level 0.7439.

Case 3: FS2 is over zone fed by TPS-02. AT7 and $FS2_{2\ up}$ *have faults.*

Status information from the SCADA system (in time order): primary backup protections of feeder lines in SP2 operated, CB_{42} tripped; meanwhile, CB_{51} tripped, AT9 auto switched over; remote backup protection $FS3_s$ of feeder lines in TSS-02 operated, CB_{63} and CB_{64} tripped. When faults occur, current directions of I_{34} and I_{35} are positive, and current is detected only in SP2 and ATP2. In this case, main protection of feeder lines in SP2, CB_{43} and main protection of AT7

refused operation, and status information of primary backup protection of AT7 lost.

According to the SCADA data and Table 4, the WFRSN P system fault diagnosis modelS for AT7 and $FS2_2$ and their parameter matrices are established to perform WMBRA, respectively. After the reasoning, the fault confidence levels AT7 and $FS2_{up}$ are obtained, i.e., 0.6946 and 0.6123. The fault region of $FS2_{up}$ can be further identified according to the fault current detected and the WFRSN P system fault diagnosis model for fault region identification in Fig. 12, and then we get the result that $FS2_{2\ up}$ has a fault. So the autotransformer AT2 has a fault with a fault confidence level 0.6946 and the feeding section $FS2_{2\ up}$ has a fault with a fault confidence level 0.6123.

The results of *Cases 1-3* give evidence of that the proposed fault diagnosis approach can obtain satisfying results both in the situation in normal power supply and over zone feeding with complete/incomplete alarm information. In addition, the proposed method can obtain the satisfying result as that in [23] by using only once simple reasoning while the method in [23] needs a second reasoning.

6 Conclusions

In this study, WFRSN P systems are applied in fault diagnosis of TPSSs and WMBRA is proposed to perform weighted matrix-based reasoning to obtain a fault confidence level for each candidate faulty section. The definitions of neurons in the WFRSN P system proposed in [18] are extended and more types of neurons are considered to express different types of status information of protection obtained form SCADA systems. Building processes and parameter setting of fault diagnosis models for sections and fault region identification of feeding sections are described in detail. Case studies show effectiveness of the presented method in diagnosing faulty sections in TPSSs. Considering the high requirement of TPSSs for diagnosing speed, how to improve WFRSN P systems to adapt themselves to online fault diagnosis is our future work.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61170016, 61373047, 61170030), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008). The last author acknowledge the support of the project TIN 2012-3734 of the Ministerio de Economía y Competitividad of Spain.

References

1. Gh. Păun, "Computing with membranes," *J. Comput. Syst. Sci.*, 61(1), 108-143 (2000)

2. M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural P systems," *Fund. Inform.*, 71(2-3), 279-08 (2006)
3. Gh. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, "Spike train in spiking neural P systems," *Int. J. Found. Comput. Sci.*, 17(4), 975-1002 (2006)
4. H. Chen, T.-O. Ishdorj, Gh. Păun, and M. J. Pérez-Jiménez, "Handling languages with spiking neural P systems with extended rules," *Romanian J. Inform. Sci. Technol.*, 9(3), 151-162 (2006)
5. R. Freund, M. Ionescu, and M. Oswald, "Extended spiking neural P systems with decaying spikes and/or total spiking," *Int. J. Found. Comput. Sci.*, 19(5), 1223-1234 (2008)
6. M. Cavaliere, O.H. Ibarra, Gh. Păun, O. Egecioglu, M. Ionescu, and S. Woodworth, "Asynchronous spiking neural P systems," *Theor. Comput. Sci.*, 410(24-25), 2352-2364 (2009)
7. L. Q. Pan and Gh. Păun, "Spiking neural P systems: an improved normal form," *Theor. Comput. Sci.*, 411(6), 906-918 (2010)
8. L. Q. Pan and X. X. Zeng, "Small universal spiking neural P systems working in exhaustive mode," *IEEE Trans. on Nanobiosci.*, 10(2), 99-105 (2011)
9. L. Q. Pan, Gh. Păun, and M. J. Pérez-Jiménez, "Spiking neural P systems with neuron division and budding," *Sci. China. Infrom. Sci.*, 58(8), 1596-1607 (2011)
10. X. Y. Zhang, B. Luo, X. Y. Fang and L. Q. Pan, " Sequential spiking neural P systems with exhaustive use of rules," *BioSystems*, 108: 52-62 (2012)
11. F. George, C. Cabarle, H. N. Adorna, M. A. Martínez, and M. J. Pérez-Jiménez, "Improving GPU simulations of spiking neural P systems," *Rom. J. Inf. Sci. Tech.*, 15(1), 5-20 (2012)
12. G. C. Francis and N. A. Henry, "On structures and behaviors of spiking neural P systems and petri nets," *Int. Conf. on Membrane Computing*, pp. 145-160 (2012)
13. T. Song, L. Q. Pan and Gh. Păun, "Asynchronous spiking neural P systems with local synchronization," *Inform. Sciences*, 219, 197-207 (2013)
14. H. Peng, J. Wang, M. J. Pérez-Jiménez, H. Wang, J. Shao, and T. Wang, "Fuzzy reasoning spiking neural P system for fault diagnosis," *Inform. Sciences*, 235, 106-116 (2013)
15. G. J. Xiong, D. Y. Shi, L. Zhu, and X. Z. Duan, "A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems," *Math. Probl. Eng.*, http://dx.doi.org/10.1155/2013/815352, 2013.
16. J. Wang and H. Peng, "Adaptive fuzzy spiking neural P systems for fuzzy inference and learning," *Int. J. Comput. Math.*, 90(4), 857-868 (2013)
17. M. Tu, J. Wang, H. Peng, and P. Shi, "Application of adaptive fuzzy spiking neural P systems in fault diagnosis of power systems," *Chinese J. Electron*, 23(1), 87-92 (2014)
18. J. Wang, P. Shi, H. Peng, Mario J. Pérez-Jiménez, and Tao Wang, "Weighted fuzzy spiking neural P system," *IEEE Trans. Fuzzy Syst.*, 21(2), 209-220 (2013)
19. G. X. Zhang, H. N. Rong, F. Neri and Mario J. Pérez-Jiménez, "An optimization spiking neural P system for approximately solving combinatorial optimization problems," *Int. J. Neural Syst.*, 24(5), 1-15 (2014)
20. S. F. Xie and Q. Z. li, "Application of expert system based on mixing reasoning in traction substation fault diagnosis," *in Proc. of IWADS*, China, pp. 229-232 (2002)
21. Y. Du, P. C. Zhan and W. Y. Yu, "A susstation fault diagnosis sysytem based on case-based reasoning and rule-based reasoing," *Power System Technology*, 28(1), 34-37 (2004)

22. R. Wang, X. C. Chen, S. B. Gao, and H. Q. Jin, "Study on the key problems of the power supply automation system for railway passenger dedicated lines," *J. China Railw. Soc.*, 28(3), 116-119 (2009)
23. S. Wu, Z. Y. He, C. H. Qian, and T. L. Zang, "Application of fuzzy petri net in fault diagnosis of tranction power supply system for high-speed way," *Power System Technology*, 35(9) 79-85, (2011)
24. W. X. Guo, F. S. Wen, G. Ledwich, Z. W. Liao, X. Z. He and J. H. Liang, "An analytic model for fault diagnosis in power systems condidering malfunctions of protective relays and circuit breakers," *IEEE Trans. on Power Deliver.*, 25(3), 1393-1401 (2010)
25. TB10621-2009, High speed railway design criterion of China.
26. Z. Q. Han, S. P. Liu, S. B. Gao, and Z. Q. Bo, "Preotection scheme for china high-speed railway," *10th IET on DPSP*, Manchester, pp. 1-5, (2010)
27. C. Peng, "Protection configuration and setting of high-speed railway AT traction power supply systems (M.S Degree Thesis)," SouthWest JiaoTong University, Chengdu, China, 2009.
28. S. B. Gao, "Study on novle protective schemes of traction power supply systems for high speed railways (Ph.D. Thesis)," SouthWest JiaoTong University, Chengdu, China, 2004.
29. J. W. Yang, Z. Y. He, and T. L. Zang, "Power system fault-diagnosis method based on directional weighted fuzzy Petri nets," *Proc. of the CSEE*, 30(34), 42-49 (2010)

Author Index