RGNC REPORT 1/2010

Eighth Brainstorming Week on Membrane Computing

Sevilla, February 1-5, 2010

Miguel Ángel Martínez del Amor
Gheorghe Păun
Ignacio Pérez Hurtado de Mendoza
Agustín Riscos Núñez
Editors

Research Group on
Natural Computing
REPORTS

UNIVERSIDAD DE SEVILLA

Eighth Brainstorming Week on Membrane Computing

Sevilla, February 1-5, 2010

Miguel Ángel Martínez del Amor
Gheorghe Păun
Ignacio Pérez Hurtado de Mendoza
Agustín Riscos Núñez
Editors

Eighth Brainstorming Week on Membrane Computing

Sevilla, February 1-5, 2010

Miguel Ángel Martínez del Amor
Gheorghe Păun
Ignacio Pérez Hurtado de Mendoza
Agustín Riscos Núñez
Editors

RGNC REPORT 1/2010

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2010

©Autores
ISBN: 978-84-614-2357-6
Depósito Legal:
Edita: Fénix Editora
41004 Sevilla
Telf: 954 90 74 36 – Móvil: 620 98 36 64
Web: www.fenixeditora.com

Preface

These proceedings contain the papers emerged from the Eighth Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from February 1 to February 5, 2010, in the organization of the Research Group on Natural Computing from the Department of Computer Science and Artificial Intelligence of Sevilla University. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and the next six editions took place in Sevilla at the beginning of February 2004, 2005, 2006, 2007, 2008, and 2009, respectively.

In the style of previous meetings in this series, the eighth BWMC was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation. Already last year the number of presentations had the tendency to increase, and this was true also for the 2010 edition: there were 31 talks. Actually, also the number of participants was significantly greater than in 2009 – see the list in the end of this preface. However, in the style of the of this series of meetings, these presentations were "provocative", mainly proposing new ideas, open problems, research topics, results which need further improvements. The efficiency of this type of meetings was again proved to be very high and the present volume prove 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 the papers from these volumes will be considered for publication in a special issues of *Romanian Journal of Information Science and Technology* (published by the Romanian Academy). After the first BWMC, a special issue of *Natural Computing* – volume 2, number 3, 2003, and a special issue of *New Generation Computing* – volume 22, number 4, 2004, were published; papers from the second BWMC have appeared in a special issue of *Journal of Universal Com-*

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

***

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

1. Alhazov Artiom, Hiroshima University, Japan,
   `aartiom@yahoo.com`- tele-participation
2. Campora Daniel Hugo, University of Sevilla, Spain
   `danielcampora@gmail.com`
3. Castellanos Juan, Polytechnical University of Madrid, Spain
   `jcastellanos@fi.upm.es`
4. Cecilia Canales José María, University of Murcia, Spain
   `chema@ditec.um.es`
5. Cienciala Ludek, Silesian University, Opava, Czech Republic
   `ludek.cienciala@fpf.slu.cz`
6. Ciencialova Lucie, Silesian University, Opava, Czech Republic
   `lucie.ciencialova@fpf.slu.cz`
7. Cobeña Morián Marcos, University of Sevilla, Spain,
   `marcobmor@alum.us.es`
8. Colomer-Cugat M. Angels, University of Lleida, Spain,
   `Colomer@matematica.UdL.es`
9. Cordón-Franco Andrés, University of Sevilla, Spain,
   `acordon@us.es`
10. Csuhaj-Varjú Erzsébet, Hungarian Academy of Sciences, Budapest, Hungary,
    `csuhaj@sztaki.hu`
11. Díaz-Pernil Daniel, University of Sevilla, Spain,
    `sbdani@us.es`
12. Escuela Gabi, Friedrich Schiller University Jena, Germany,
    `gabi.escuela@uni-jena.de`

13. Fernández Márquez, Carlos, University of Sevilla, Spain,
carfermar@alum.us.es
14. Ferretti Claudio, University of Milano-Bicocca, Italy,
ferretti@disco.unimib.it
15. Freund Rudolf, Technical University Wien, Austria,
rudi@emcc.at, rudi@logic.at
16. García-Quismondo Manuel, University of Sevilla, Spain
mangarfer2@alum.us.es
17. Gheorghe Marian, University of Sheffield, United Kingdom,
marian@dcs.shef.ac.uk
18. Graciani-Díaz Carmen, University of Sevilla, Spain,
cgdiaz@us.es
19. Gutiérrez-Naranjo Miguel Ángel, University of Sevilla, Spain,
magutier@us.es
20. Ipate Florentin Eugen, University of Pitesti, Romania,
florentin.ipate@ifsoft.ro
21. Kelemen Josef, Silesian University, Opava, Czech Republic,
kelemen@fpf.slu.cz
22. Kogler Marian, Technical University Wien, Austria,
marian@emcc.at
23. Krassovitskiy Alexander, Rovira i Virgili University, Tarragona, Spain,
alexander.krassovitskiy@estudiants.urv.cat
24. Lefticaru Raluca, University of Pitesti, Romania,
raluca.lefticaru@gmail.com
25. Leporati Alberto, University of Milano-Bicocca, Italy,
leporati@disco.unimib.it
26. Lombardo Rosario, University of Verona, Italy,
rosario.lombardo@univr.it
27. Maggiolo Andrea, University of Pisa, Italy,
maggiolo@di.unipi.it
28. Marchena Blanco Virginia, University of Sevilla, Spain,
virmar@gmail.com
29. Marchetti Luca, University of Verona, Italy,
luca.marchetti@univr.it
30. Martínez-del-Amor Miguel Ángel, University of Sevilla, Spain,
mdelamor@us.es
31. Mauri Giancarlo, University of Milano-Bicocca, Italy,
mauri@disco.unimib.it
32. Milazzo Paolo, University of Pisa, Italy,
milazzo@di.unipi.it
33. Murphy Niall, NUI Maynooth, Ireland
nmurphy@cs.nuim.ie
34. Nicolescu Radu, University of Auckland, New Zealand
r.nicolescu@auckland.ac.nz

35. Obtułowicz Adam, Polish Academy of Sciences, Poland,
A.Obtulowicz@impan.gov.pl
36. Orejuela-Pulido Enrique Francisco, University of Sevilla, Spain
enrique.orejuela@gmail.com
37. Pagliarini Roberto, University of Verona, Italy,
roberto.pagliarini@univr.it
38. Păun Gheorghe, Institute of Mathematics of the Romanian Academy, Bucharest, Romania, and University of Sevilla, Spain,
george.paun@imar.ro, gpaun@us.es
39. Peña Camacho Miguel Ángel, Polytechnical University of Madrid, Spain,
mapc@eui.upm.es
40. Pérez-Hurtado-de-Mendoza Ignacio, University of Sevilla, Spain,
perezh@us.es
41. Pérez-Jiménez Mario de Jesús, University of Sevilla, Spain,
marper@us.es
42. Porreca Antonio, University of Milano-Bicocca, Italy,
porreca@disco.unimib.it
43. Ramírez-Martínez Daniel, University of Sevilla, Spain,
thebluebishop@gmail.com
44. Riscos-Núñez Agustín, University of Sevilla, Spain,
ariscosn@us.es
45. Rodríguez-López Raquel, University of Sevilla, Spain,
raqrodlop@alum.us.es
46. Rogozhin Yurii, Institute of Mathematics and Computer Science, Chisinau, Moldova,
rogozhin@math.md
47. Romero-Jiménez Alvaro, University of Sevilla, Spain,
romero.alvaro@us.es
48. Sánchez Canales Vanesa, University of Sevilla, Spain,
vscanales@gmail.com
49. Sburlan Dragoş, Ovidius University, Constanţa, Romania,
dsburlan@univ-ovidius.ro
50. Sempere Luna José María, Polytechnical University of Valencia, Spain,
jsempere@dsic.upv.es
51. Stefanescu Gheorghe, University of Bucharest, Romania,
gheorghe@funinf.cs.unibuc.ro
52. Tini Simone, University of Insubria, Italy,
simone.tini@uninsubria.it
53. Troina Angelo, University of Torino, Italy,
troina@di.unito.it
54. Vaszil György, Hungarian Academy of Sciences, Budapest, Hungary,
vaszil@sztaki.hu
55. Zandron Claudio, University of Milano-Bicocca, Italy,
zandron@disco.unimib.it

56. Zeng Xiangxiang, Huazhong University of Science and Technology, Wuhan, R.P. China,
    `xzeng@foxmail.com`

As mentioned above, the meeting was organized by the Research Group on Natural Computing (PAI TIC-193) from Sevilla University (`http://www.gcn.us.es`)– and all the members of this group were enthusiastically involved in this (not always easy) work. The meeting was supported from various sources: (i) Proyecto de Excelencia con investigador de reconocida valía, de la Junta de Andalucía, grant P08 – TIC 04200, (ii) Proyecto del Ministerio de Ciencia e Innovación, grant TIN2009 – 13192, (iii) Red Temática en Computación Biomolecular y biocelular, grant TIN2008-04487-E, (iv) IV Plan Propio de la Universidad de Sevilla, well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Mario de Jesús Pérez-Jiménez
(Sevilla, April 30, 2010)

Contents

On Communication Complexity in Evolution-Communication P Systems

Henry Adorna[1], Gheorghe Păun[2,3], Mario J. Pérez-Jiménez[3]

[1] Department of Computer Science (Algorithms and Complexity)
University of the Philippines-Diliman
1101 Quezon City, Philippines
hnadorna@dcs.upd.edu.ph

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania

[3] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: gpaun@us.es, marper@us.es

Summary. Looking for a theory of communication complexity for P systems, we consider here so-called evolution-communication (EC for short) P systems, where objects evolve by multiset rewriting rules without target commands and pass through membranes by means of symport/antiport rules. (Actually, in most cases below we use only symport rules.) We first propose a way to measure the communication costs by means of "quanta of energy" (produced by evolution rules and) consumed by communication rules. EC P systems with such costs are proved to be Turing complete in all three cases with respect to the relation between evolution and communication operations: priority of communication, mixing the rules without priority for any type, priority of evolution (with the cost of communication increasing in this ordering in the universality proofs).

More appropriate measures of communication complexity are then defined, as dynamical parameters, counting the communication steps or the number (and the weight) of communication rules used during a computation. Such parameters can be used in three ways: as *properties* of P systems (considering the families of sets of numbers generated by systems with a given communication complexity), as *conditions* to be imposed on computations (accepting only those computations with a communication complexity bounded by a given threshold), and as standard *complexity measures* (defining the class of problems which can be solved by P systems with a bounded complexity). Because we ignore the evolution steps, in all three cases it makes sense to consider hierarchies starting with finite complexity thresholds. We only give some preliminary results about these hierarchies (for instance, proving that already their lower levels contain complex – e.g., non-semilinear – sets), and we leave open many problems and research issues.

1 Introduction

Although membrane computing can still be considered as young branch of natural computing, [11], the theory of P systems is well developed (see details in [15] and at the area website from [19]). In particular, many research efforts were devoted to complexity issues related to P systems. First, all results dealing with the number of membranes, with normal forms, with various static parameters (estimating the size of P systems from various points of view) can be considered as contributing to the *descriptional complexity* of P systems (this was a much investigated topic in formal languages theory, see, e.g., [5]). Much more coherently developed, forming an explicit theory, is the *computational complexity* of P systems, where the main complexity parameter is *time*, the number of steps of a computation. In this framework, several specific complexity classes were defined, elaborate relations with the $\mathbf{P} \neq \mathbf{NP}$ conjecture were found, as well as many other related results; we refer to the recent survey [17] and to the corresponding chapter of [15] for details. Recently, also a *space* complexity theory for P systems was initiated – see, e.g., [18].

What is still almost not at all considered is the *communication complexity* of P systems, in spite of the fact that this was suggested as a research topic several times (for instance, in [3] and [13], but without any hint about a possible definition). A related measure is discussed in [4] for symport/antiport P systems, the so-called "communication difference", denoted $Comdif$, defined as the difference of numbers of input and output objects in an antiport rule, but this is again a static measure, defined for rules of a P system.

We address this issue here, with several basic proposals, but our preliminary results indicate that this is a non-trivial research direction – especially if we want to get close to the classic theory of communication complexity, as synthesized, e.g., in [7]. Roughly speaking, the classic framework deals with a distributed/parallel computing device and complex problems which are split into subproblems and the parts are distributed to separate "processors", which cooperate in solving the general problem; to this aim, the processors need to communicate and the number of bits used to this aim gives the communication complexity of the solution. P systems are distributed and parallel devices, but we do not have an explicit protocol for solving problems in a distributed manner, after introducing subproblems of a problem in various subsystems; we have various tools for communication among membranes, but the amount of communication was not yet explicitly and systematically investigated.

The present paper mainly calls once again the attention to this research direction, as we do not propose yet a way to solve problems in a distributed way using a P system, so that a framework as that in [7] to be obtained. Instead, we first go back to a sort of a descriptional complexity measure, defined for evolution-communication (EC for short) P systems as introduced in [1] which was then investigated in a series of papers, e.g., in [8]. In these systems, the evolution of objects is separated from their communication across membranes: the evolution is done by means of multiset rewriting rules and the communication by symport/antiport rules. In order to evaluate the communication effort, we consider a *cost* of using a communication rule, in the form of a quantity of “energy” consumed by that rule. Specifically, we consider a special object, e, called “quantum of energy”; evolution rules can produce amounts of energy, which is consumed by the communication rules. Three types of EC P systems are investigated: with priority of communication on evolution, with steps where rules of the two types are non-deterministically mixed, and with priority of evolution on communication. In all cases, universality results are obtained (this was known only for the intermediate case, of no priority among the two types of operations), but, interesting enough, the cost of communication increases in the order we have mentioned the three cases: one quantum per rule, three quanta per rule, and five quanta per rule, respectively, are used in the corresponding proofs.

We note that P systems with the use of rules controlled by energy were considered already, e.g., in [16], [9], [10], but in different frameworks (other types of P systems, different goals).

A natural step is then to pass from this kind of a static measure to some dynamical ones, with natural definitions: count the communication steps, or the communication rules, or all quanta of energy used during all steps of a halting computation. An infinite cost (hence infinitely many communication rules used during the computation) leads to universality, but also considering only a finite number of communication steps (of communication rules used during a computation) makes sense. Actually, counting only the number of communication steps is nothing else than the length of the computation (the time complexity) . . . ignoring the evolution steps. This means that the computation can be arbitrarily complex in terms of evolution steps (we however use here only non-cooperative evolution rules), what is counted are the communication rules. Such a parameter can be investigated from three points of view: as a *property* of P systems, as a *condition* for selecting only certain computations as acceptable, as the *effort* to solve (decision) problems. In all cases, we can start with finite thresholds (no communication step, one communication step, and so on), hence infinite hierarchies are expected. We only give some hints about the possible proofs of the infinity of these hierarchies, as well as examples showing that already the lower levels of the hierarchies contain complex sets of numbers (e.g., non-semilinear). Thus, besides definitions and preliminary results, we provide here mainly open problems and research topics.

2 Some Prerequisites

Before introducing the class of P systems we investigate in this paper, let us fix some notation and terminology.

The free monoid generated by an alphabet V is denoted with V^* and its neutral element (the empty string) is denoted by λ; the set $V^* - \{\lambda\}$ (of non-empty strings over V) is denoted by V^+. For $a \in V, x \in V^*$, $|x|$ denotes the length of x, and $|x|_a$ is the number of occurrences of symbol a in the string x.

The families of semilinear and of recursively enumerable sets of vectors of dimension $k \geq 1$ of natural numbers are denoted by $SLIN^k$ and N^kRE, respectively; if $k = 1$ (hence we deal with numbers, vectors of one dimension), then the superscript is omitted.

In the proofs of our universality results we use the notion of a (non-deterministic) *register machine*, which is a device $M = (m, B, l_0, l_h, R)$, where $m \geq 1$ is the number of registers, B is the (finite) set of instruction labels, l_0 is the initial label, l_h is the halting label, and R is the finite set of instructions labeled (uniquely identified, with each label being associated with only one instruction) by elements from B. The labeled instructions are of the following forms:

– $l_i : (\mathtt{ADD}(r), l_j, l_k)$, $1 \leq r \leq m$ (add 1 to register r and go nondeterministically to one of the instructions with labels l_j, l_k),
– $l_i : (\mathtt{SUB}(r), l_j, l_k)$, $1 \leq r \leq m$ (if register r is not empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k).

A register machine generates a set of natural numbers in the following manner: we start computing with all m registers empty, with the instruction labeled l_0; if the label l_h is reached, then the computation *halts* and the value of register 1 is the number generated by the computation (without loss of generality, all other registers can be assumed to be empty at that time). The set of all natural numbers generated in this way by M is denoted by $N(M)$. It is known that non-deterministic counter machines (with three counters) can generate any set of Turing computable sets of natural numbers. If the contents of several registers is taken into consideration in the end of a computation, then vectors of natural numbers are generated.

Convention: When comparing two number generating devices, number 0 is omitted.

3 Evolution-Communication P Systems

Although the notions we work with are introduced below, it would be useful if the reader has some familiarity with basic facts of membrane computing.

The class of P systems which we investigate in this paper is that of EC P systems introduced in [1]. Such a system is a construct of the form

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, R'_1, \ldots, R_m, R'_m, i_{out}),$$

where O is the alphabet of objects, μ is the membrane structure (with m membranes, organized in a hierarchical manner, hence with the membrane structure described by a tree, and given as an expression of labeled parentheses; in this definition, like in most cases in the paper, the membranes are labeled with natural numbers, but any alphabet of labels may also be used), $w_1, \ldots, w_m$ 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, $R'_1, \ldots, R'_m$ are finite sets of symport/antiport rules associated with the membranes of μ, and i_{out} is the label of the output membrane. (Note that the evolution rules are associated with the regions and the communication rules are associated with the membranes.) The number m of membranes in μ is called the *degree* of Π. The evolution rules are of the form $a \to v$, where $a \in O, v \in O^*$ (hence non-cooperative and without target indications in the right hand side, as usual in transition P systems), while the symport/antiport rules are of the standard forms used in symport/antiport P systems ($(u, in), (u, out)$ for symport rules and $(u, out; v, in)$ for antiport rules, where $u, v \in O^+$; the length of u in a symport rule and the maximum of $|u|, |v|$ in an antiport rule is called the *weight* of the rule, with the maximum degree over all rules being called the weight of the system).

The weight of symport and antiport rules gives already an indication on the communication complexity of the system. Here we stress this, in the following way. We allow only minimal symport and antiport rules, hence with the multisets u, v above consisting of single objects in O, and, moreover, we add to the system a special object, e, which does not belong to O and can appear in rules as follows. The evolution rules can be of the form $a \to v$, with $a \in O, v \in (O \cup \{e\})^*$, and the symport/antiport rules can be of the forms $(ae^i, in), (ae^i, out)$, where $a \in O, i \geq 1$, and $(ae^i, out; be^j, in)$, where $a, b \in O$ and $i, j \geq 0, i+j \geq 1$. Note that the "quantum of energy" e can be produced by evolution rules and that no communication can be done without involving an amount of "energy". Actually, this energy is consumed by the communication rules: after using a symport or antiport rule, the objects of O are transported across the membrane with which the rule is associated and the occurrences of object e are lost, they do not pass from a region to another one. In the proofs from the next section we will discuss in some detail the way the communication rules are applied, so we do not give here any example.

In a symport rule as above, the number i is called the *energy* of the rule, while in an antiport rule as above the sum $i + j$ is called the *energy*.

In [1] (and [8]), the rules of an EC P systems are used in a non-deterministic maximally parallel way, without any separation of evolution and communication operations. Here we consider three cases (we call them *modes*): (i) communication has priority over evolution (indicated by CPE) – if a communication rule is applicable in any membrane of the system, then at this step only communication rules are used (in the non-deterministic maximally parallel way), and no evolution rule is applied in the system; (ii) communication and evolution rules are applied together, mixed (indicated by CME), as in [1]; (iii) evolution has priority over communication (indicated by EPC) – if any evolution rule can be used in the system, then only such rules are used at that step, and no communication operation is performed.

Using the rules in the non-deterministic maximally parallel way, in one of the three modes suggested above, we obtain transitions among configurations of the system, then computations and halting computations as usual in membrane computing. In the next section we consider the set $N_{mode}(\Pi)$ of numbers generated by a P system Π, by counting the objects of O present in region i_{out} in the halting configuration of computations in Π, performed in the in the mode

$mode \in \{CPE, CME, EPC\}$. The family of sets of numbers $N_{mode}(\Pi)$ generated in this way by EC P systems with at most $m \geq 1$ membranes, symport rules of maximal energy $p \geq 0$ and antiport rules of maximal energy $q \geq 0$, is denoted by $N_{mode}ECP_m(sym_p, anti_q)$; when one of the parameters m, p, q is not bounded, we replace the respective subscript with $*$.

4 The Power of EC P Systems with Energy

We start by pointing out some inclusions which directly follow from the definitions:

Lemma 1. *$N_{mode}ECP_m(sym_p, anti_q) \subseteq N_{mode}ECP_{m'}(sym_{p'}, anti_{q'}) \subseteq NRE$, for all $1 \leq m \leq m'$, $0 \leq p \leq p'$, $0 \leq q \leq q'$, and $mode \in \{CPE, CME, EPC\}$; each of m', p', q' can also be equal to $*$.*

Actually, most of the inclusions above are equalities.

Theorem 1. *$N_{CPE}ECP_m(sym_p, anti_q) = NRE$, for all $m \geq 4, p \geq 1, q \geq 0$.*

Proof. We only prove the inclusion $NRE \subseteq N_{CPE}ECP_4(sym_1, anti_0)$, and to this aim we use the characterization of NRE by means of register machines with three registers. Let $M = (3, B, l_0, l_h, R)$ be such a machine. We construct an EC P system

$$\Pi = (O, e, [_0[_1\ ]_1[_2\ ]_2[_3\ ]_3\ ]_0, w_0, w_1, w_2, w_3, R_0, R'_0, R_1, R'_1, R_2, R'_2, R_3, R'_3, 1),$$

with

$$O = \{l, l', l'', l''', \bar{l} \mid l \in B\} \cup \{a, \#\},$$
$$w_0 = l'_0,\ w_1 = w_2 = w_3 = \lambda,$$

and the sets of rules consist of the rules mentioned in the following tables, which show the way the system Π simulates the instructions of M (note that initially we have the primed version of the initial label of M present in the skin region and nothing else in the whole system).

For any ADD instruction of the form l_i : $(\mathtt{ADD}(r), l_j, l_k)$ in R, we perform the following five steps in Π (we indicate for each step the rules associated with membranes/regions 0 and r):

Step	R_0	R_r	R'_r
1	$l'_i \to l_i e$	–	–
2	–	–	$(l_i e, in)$
3	–	$l_i \to \bar{l}_i a e$	–
4	–	–	$(\bar{l}_i e, out)$
5	$\bar{l}_i \to l'_s, s = j, k$	–	–

We also add the rules

$$\bar{l}_i \to \#,\ \# \to \#$$

to R_r (they are used in steps 5 and 6 (and then $\# \to \#$ forever) if in step 4 one uses the rule (ae, out) which is present in R'_r if there are SUB instructions for this register – see below).

The simulation of the ADD rule is obvious – and the fact that the communication has priority over evolution plays no role here as only one (type of) rule can be applied in any step.

The simulation of a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in R is more intricate. We first indicate the rules present in sets R_0, R_r, R'_r:

$$\begin{aligned}
R_0 : {} & l'_i \to l_i e, \\
& l'''_i \to l'_j, \\
& l''_i \to l'_k, \\
R_r : {} & l_i \to l'_i e, \\
& l'_i \to l''_i, \\
& l''_i \to l'''_i e, \\
R'_r : {} & (l_i e, in), \\
& (ae, out), \\
& (l'''_i e, out), \\
& (l''_i e, out).
\end{aligned}$$

The way these rules are used for simulating the SUB instruction in the case when the register r is non-empty is shown in the next table:

Step	R_0	R_r	R'_r
1	$l'_i \to l_i e$	–	–
2	–	–	$(l_i e, in)$
3	–	$l_i \to l'_i e$	–
4	–	–	(ae, out)
5	–	$l'_i \to l''_i$	–
6	–	$l''_i \to l'''_i e$	–
7	–	–	$(l'''_i e, out)$
8	$l'''_i \to l'_j$	–	–

The simulation of the SUB instruction in the case of an empty register r is indicated below:

Step	R_0	R_r	R'_r
1	$l'_i \to l_i e$	–	–
2	–	–	$(l_i e, in)$
3	–	$l_i \to l'_i e$	–
4	–	$l'_i \to l''_i$	–
5	–	–	$(l''_i e, out)$
6	$l''_i \to l'_k$	–	–

The first three steps are identical. If register r is non-empty, hence membrane r contains objects a, then, because the communication has priority, one copy of a is removed from membrane r and the rule $l_i' \to l_i''$ is used in the next step. If the register is empty, then the rule $l_i' \to l_i''$ is used already in step 3; because the copy of e remained unused in membrane r, now l_i'' can exit, and it introduces l_k' in the skin region. If e was consumed in step 3, then l_i'' can evolve to l_i''' and only now it can leave membrane r, introducing l_j' in the skin region.

The simulation of the SUB instruction is correct. The simulation of ADD and SUB instructions can continue until introducing the object l_h', which cannot evolve, the computation stops. The contents of membrane 1 corresponds to the contents of register 1, hence $N(M) = N_{CPE}(\Pi)$; the observation that we use only symport rules with energy 1 completes the proof.

The priority of communication is a useful "programming tool"; in its absence, the system has to use more energy quanta.

Theorem 2. *$N_{CME}ECP_m(sym_p, anti_q) = NRE$, for all $m \geq 4, p \geq 3, q \geq 0$.*

Proof. Consider a set $Q \in NRE$ and take $Q' = \{n-1 \mid n \in Q\}$. If $1 \notin Q$, then we proceed as follows. We take a register machines with three registers, $M = (3, B, l_0, l_h, R)$, generating the set Q' and we construct the EC P system

$$\Pi = (O, e, [_0[_1\ \ ]_1[_2\ \ ]_2[_3\ \ ]_3\ ]_0, w_0, w_1, w_2, w_3, R_0, R_0', R_1, R_1', R_2, R_2', R_3, R_3', 1),$$

with

$$O = \{l, l', l'', l''', \bar{l} \mid l \in B\} \cup \{a, \#\},$$
$$w_0 = l_0',\ w_1 = w_2 = w_3 = \#,$$

and the sets of rules constructed as indicated below.

For any ADD instruction of M we use the same rules as in the proof of the previous theorem.

For a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in R we consider the following rules:

$$\begin{aligned}
R_0 : & \ l_i' \to l_i e,\\
& \ l_i''' \to l_j',\\
& \ l_i'' \to l_k',\\
& \ \# \to \#,\\
R_r : & \ l_i \to l_i' e^2,\\
& \ l_i' \to l_i'' e,\\
& \ l_i'' \to l_i''',\\
R_r' : & \ (l_i e, in),\\
& \ (a e^2, out),\\
& \ (l_i'' e^3, out),\\
& \ (l_i''' e, out),\\
& \ (\# e^3, out).
\end{aligned}$$

The simulation of the SUB instruction proceeds as follows. Object l_i enters membrane r, produces here two quanta of energy and passes to l_i'. Because of the maximal parallelism, if register r is not empty, in the next step both rules $l_i' \rightarrow l_i''e$ and (ae^2, out) are used, hence we have to pass further to l_i''', which, in the next step, exits to the skin membrane, where it introduces l_j'. If the register r is empty, then no rule can use the two quanta of energy introduced by the rule $l_i \rightarrow l_i'e^2$. By using the rule $l_i' \rightarrow l_i''e$, in step 4 one further occurrence of e is introduced in membrane r. In step 5, three rules can be used: $l_i'' \rightarrow l_i''', (l_i''e^3, out)$, and $(\#e^3, out)$; if the first rule is used, then the trap-object # exits the membrane and the computation never halts, because of the rule $\# \rightarrow \#$ from R_0. Thus, the only continuation which can lead to a halting computation is to use the rule $(l_i''e^3, out)$, and thus l_k' is introduced in the skin region. In both cases, the simulation of the SUB instruction is correct, hence we obtain $N_{CME}(\Pi) = Q$: in membrane 1, besides copies of object a corresponding to the value of register 1 in the halting configuration of M, we also have a copy of object #.

If the set Q contains the number 1, then Q' contains 0 instead, which is ignored when considering a register machine for Q'. However, number 1 can be generated separately: consider an additional object, b, present initially in the skin membrane, and the rules $b \rightarrow \lambda$ and $b \rightarrow l_0'$. In the first case, the computation stops with only one object in membrane 1, in the later case we start simulating the computations in the register machine M which generates Q'.

We conclude the proof by observing that the system Π uses symport rules with maximal energy equal to 3.

Somewhat surprising, the priority of the evolution rules over the communication rules seems to be a weaker feature than using the communication rules with priority, and this can be "explained" by the fact that communication moves objects from a region to another one, hence changes the rules to be applied to the moved objects; in general, the localization (of objects and of rules) is known to be a powerful feature of P systems. As a consequence, a larger number of membranes and amount of energy is needed in this case.

Theorem 3. *$N_{EPC}ECP_m(sym_p, anti_q) = NRE$, for all $m \geq 7, p \geq 5, q \geq 0$.*

Proof. We start again from a register machine $M = (3, B, l_0, l_h, R)$ and construct an EC P system Π simulating it. This time, the components of Π are indicated in a graphical form, in Figure 1. We explicitly mention the rules associated with an ADD instruction and a SUB instruction operating on register r; specifically, the rules of Π which simulate the ADD instruction are written in the left hand of the figure, under the instruction ADD, and the rules which simulate the SUB instruction are written in the right hand of the figure. The rules associated with the other two registers – denoted in the figure with s and t (hence $\{r, s, t\} = \{1, 2, 3\}$) – are not mentioned.

The simulation of the ADD instruction is obvious, so we only explain the way a SUB instruction is simulated. Object l_i "guesses" whether or not register r is

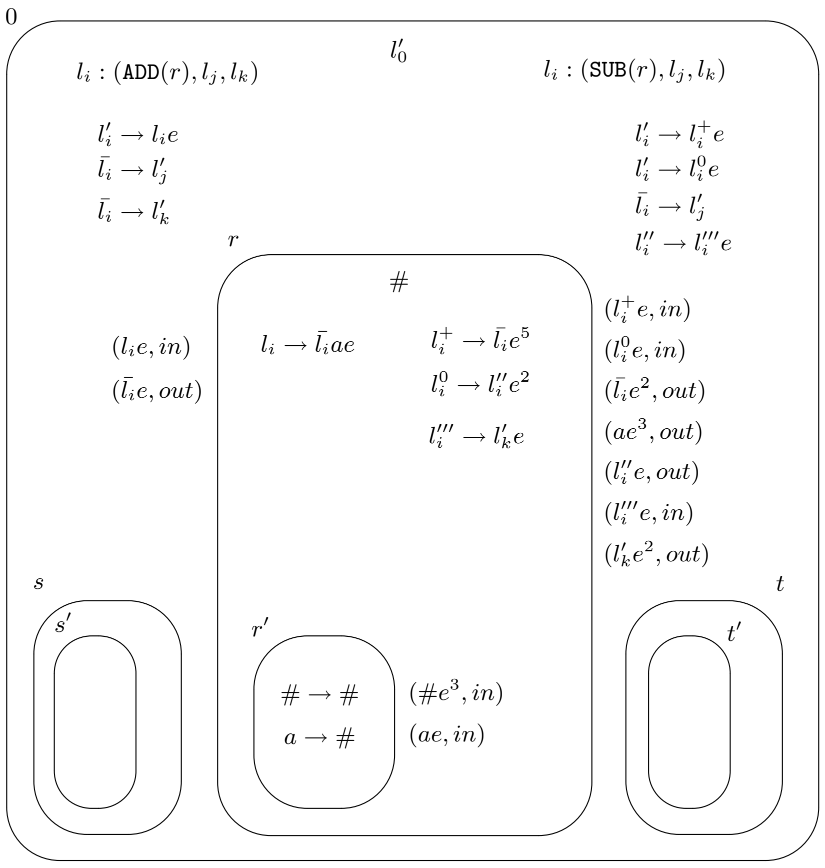


Fig. 1. The P system from the proof of Theorem 3

empty, that is it non-deterministically passes to one of l^+_i and l^0_i. Any of these objects enters membrane r. If l^+_i was produced and the register was empty, then the computation will enter an infinite cycle. Similarly, if object l^0_i was produced and the register is not empty the computation will never stop.

Let us assume that we have object l^+_i in membrane r. It produces here five quanta of energy and passes to $\bar{l}_i$ (remember that l_i uniquely labels an instruction, hence there is no ambiguity in using the bar notation, as in the simulation of the ADD instruction). No evolution rule can be used in the system, hence we are allowed (and we have) to apply communication rules. If any of the rules $(\#e^3, in), (ae, in)$ from $R'_{r'}$ are used, then the trap object $\#$ will evolve forever in membrane r'. This can be avoided only if the rules $(\bar{l}_i e^2, out), (ae^3, out)$ of R'_r are used, and this is possible if the register is non-empty; otherwise, no object a is present in membrane r, hence, because of the maximal parallelism, the rule $(\#e^3, in)$ will bring the trap-object in membrane r' and the computation never halts. If $\bar{l}_i$ arrives in the skin region, then it introduces here l'_j, which is the correct continuation of the computation in M.

Assume now that object l_i^0 was produced and introduced in membrane r. By rule $l_i^0 \to l''_i e^2$ we introduce two quanta of energy. One of them can be used by the rule $(l''_i e, out)$ from R'_r. If the membrane contains no copy of a, hence the "guess" made in the first step was correct, i.e., the register is empty, then the rule (ae, in) from $R'_{r'}$ cannot be used, otherwise the computation never stops (because of the maximal parallelism, this rule must be used in parallel with $(l''_i e, out)$; of course, the rule (ae, in) can be used even twice, with the same result, the infinite run of the computation). The copy of e waits inside membrane r until l''_i evolves in the skin region to l'''_i and this object enters membrane r. It produces here one further copy of e and exits in the form of l'_k, thus completing this branch of the simulation.

In both cases, the simulation of the instruction SUB is correct (the computation in Π never ends if an incorrect guess is made or a "wrong" rule is used). The system Π generates the same set of numbers as the register machine M, always augmented by 1: the object # remains in membrane 1 in the end of the computation. Like in the proof of Theorem 2, we can now make sure that we generate the set $Q \in NRE$ we want, number 1 included if it belongs to Q, and this completes the proof.

Several *open problems* can be formulated with respect to these three results, concerning the parameters involved in them: Can the number of membranes be decreased? Can the energy of rules in Theorems 2, 3 be decreased? Does the use of antiport rules help in this respect? (P systems with minimal symport/antiport rules are already universal – sometimes with some "garbage" objects remaining in the output membrane, see, e.g., the corresponding chapter from [15] – hence the energy can be completely removed if powerful enough communication rules are used.)

5 Dynamical Communication Complexity Measures

Let us now proceed to defining communication complexity parameters starting from computations, not from the (rules of the) system. There are (at least) three basic possibilities: (i) to count the number of steps of a computation when communication operations are done, (ii) to count the total number of communication rules used during a computation, and (iii) to consider the sum of the weights of all communication rules used during a computation (or the energy involved in these rules, in the case of EC P systems with a cost of communication, as considered in the previous sections).

More formally, let $\delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ be a halting computation in a given EC P system Π, with w_0 being the initial configuration. We interpret/represent the configurations as strings, specifying the multisets of objects placed in the regions of a membrane structure written as a string of labeled matching parentheses, hence we can speak about the number of occurrences of a symbol in such a string/configuration. Then, for each $i = 0, 1, \ldots, h-1$ we can write:

$$\begin{aligned}
ComN(w_i \Longrightarrow w_{i+1}) &= \begin{cases} 1, \text{ if a communication rule is used in this transition,} \\ 0, \text{ otherwise} \end{cases} \\
ComR(w_i \Longrightarrow w_{i+1}) &= \text{the number of communication rules used} \\
& \text{ in this transition,} \\
ComW(w_i \Longrightarrow w_{i+1}) &= \text{the total energy of the communication rules used} \\
& \text{ in this transition.}
\end{aligned}$$

These parameters can then be extended in the natural way to computations, results of computations, systems, sets of numbers: for $ComX \in \{ComN, ComR, ComW\}$ we define

$$\begin{aligned}
ComX(\delta) &= \sum_{i=0}^{h-1} ComX(w_i \Longrightarrow w_{i+1}), \text{ for } \delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h \\
& \text{ a halting computation,} \\
ComX(n, \mathit{\Pi}) &= \min\{ComX(\delta) \mid \delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h \\
& \text{ is a halting computation in } \mathit{\Pi} \text{ with the result } n\}, \\
ComX(\mathit{\Pi}) &= \max\{ComX(n, \mathit{\Pi}) \mid n \in N(\mathit{\Pi})\}, \\
ComX(Q) &= \min\{ComX(\mathit{\Pi}) \mid Q = N(\mathit{\Pi})\}.
\end{aligned}$$

In this way, we have the possibility to assess the communication complexity of sets of numbers with respect to EC P systems; actually, the class of systems can be changed, as communication plays an important role in many classes of P systems, so that we can write $ComX_{CL}(Q)$, where CL is a particular class of P systems, and in this way, we can compare the complexity of a set of numbers with respect to various types of systems generating it.

Now, we can also consider families of sets of numbers of a given maximal complexity:

$$NF_{ComX}(k) = \{Q \subseteq \mathbf{N} \mid ComX(Q) \leq k\},$$

for given $k \geq 0$, with $NF_{ComX}(fin)$ being the union of all these families, and $NF_{ComX}(\infty)$ the family of all sets of numbers computed by P systems of a given type. As in most cases, this family is equal to NRE, while non-cooperative P systems without communication generate only semilinear sets, [12], hence we can write

$$\begin{aligned}
&SLIN \subseteq NF_{ComX}(0) \subseteq NF_{ComX}(1) \subseteq \ldots \\
&\ldots \subseteq NF_{ComX}(fin) \subseteq NF_{ComX}(\infty) = NRE.
\end{aligned}$$

Many open problems and research topics arise in this context. First, the previous definition refers to EC P systems (hence to symport/antiport communication rules), but not to the way of using the rules, in the sense of the relation between evolution and communication operations – remember that in the previous sections we distinguished three possibilities in this respect.

This can be very important for the results we obtain: let us consider the system Π whose initial configuration is indicated in Figure 2, and define transitions in the mode CPE, hence with priority of communication over evolution (the output membrane is the skin one).

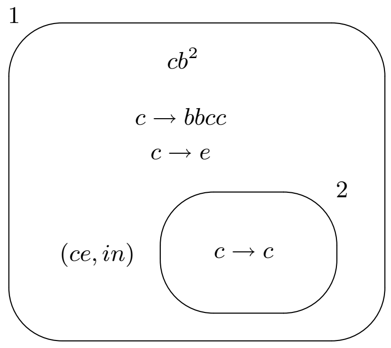


Fig. 2. An EC P system generating a non-semilinear set of numbers

Using the rule $c \rightarrow bbcc$, the number of copies of b and c increases exponentially. The number of objects b will always be twice the number of objects c in membrane 1. If in a given step we use both the rule $c \rightarrow bbcc$ and the rule $c \rightarrow e$, then we have at the same time both objects c and e in membrane 1. Hence the rule (ce, in) can be used, with priority. In this case, the computation never stop, since the rule $c \rightarrow c$ can be used forever in membrane 2. Therefore, we either use only the rule $c \rightarrow bbcc$ or only the rule $c \rightarrow e$. After n finite number of steps, the computation will stop with $3 \cdot 2^n$ objects in region 1, for some $n \geq 0$ (the value $n = 0$ is obtained if we use the rule $c \rightarrow e$ in the first step). This means that the system generates a non-semilinear set of numbers.

If we go back to our definition, we can extend it to other classes of P systems. A direct passage is via P systems with active membranes, where we have *in* and *out* rules much similar to the symport rules (but used in a sequential way, one in each membrane). Also transition P systems have communication commands, which can be counted when defining communication complexity parameters as above.

Then, more technically, we ask the folowing: is the previous hierarchy of complexity classes infinite? We conjecture that for many types of P systems (we believe that this is the case for EC P systems, with or without energy associated with communication rules, without a priority relation between evolution and communication operations) we have the following relations:

$$SLIN = NF_{ComX}(k) = NF_{ComX}(fin) \subset NF_{ComX}(\infty) = NRE,$$

for all $k \geq 0$. Interesting enough intrinsically, such a result would make non-interesting (showing that they are not sensitive enough) the complexity measures themselves. Since only two complexity classes are distinguished, sets of finite complexity and sets of infinite complexity. (This should be contrasted, for instance, with the results related to the index of context-free languages – see details in [5] – which have a somewhat similar definition, but leads to an infinite hierarchy of languages.)

A natural extension is from sets of numbers to sets of vectors of numbers. Sometimes this makes differences between families generated.

An interesting idea is also to relate the communication complexity, defined in any given way, with the number computed, i.e., to consider measures of the form

$$ComX'(n) = \frac{ComX(n, \Pi)}{n},$$

where $n \in N(\Pi)$. Such "relativized" measures are not at all frequent in membrane computing, although the relation with (the length of) the result of a computation looks appealing.

From our definitions, it directly follows that we have the relations $ComN(\alpha) \leq ComR(\alpha) \leq ComW(\alpha)$, for all possible α – step of a computation, computation, system, etc. It would be interesting to study, e.g., the family of numbers generated by systems Π for which the previous relations are equalities.

6 Communication Complexity as a Computation Regulator

In the previous section, we have considered the "amount of communication" as a property of computations and computing devices. An attractive idea would be to change the perspective and consider (the value of) this parameter as a *condition* to be imposed to computations in a given system. Instead of considering the set of numbers computed by all computations, we consider only a subset of all such numbers which can be obtained in the end of computations with a bounded communication complexity. Specifically, we can define $N_{mode}(\Pi, ComX \leq k)$ to be the set of numbers which can be generated by an EC P system Π (with energy associated with the communication rules) by means of computations δ such that $ComX(\delta) \leq k$, for $k \geq 0$.

In what follows, we consider only the measure $ComN$ (the number of steps when communication rules are used). Like in the previous section, an example is given proving that, at least for modes CPE, CME, systems with a small communication complexity can generate already non-semilinear sets. The example is given in Figure 3. For this system Π, we have

$$N_{mode}(\Pi, ComN \leq 1) = \{2^n \mid n \geq 1\},$$

for *mode* $\in \{CPE, CME\}$ (the output membrane is the internal one).

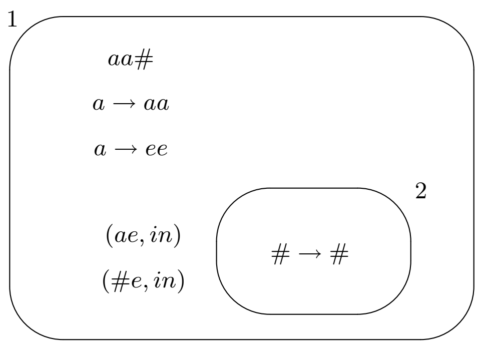


Fig. 3. An EC P systems with restricted communication complexity

It is clear from Figure 3 that the computation can stop only after using the rule $a \to ee$, and without using the rule $(\#e, in)$. If any copy of e is introduced, then a communication must be done, irrespective whether communication has priority or not (in the CME case, because of the maximal parallelism, the rule $(\#e, in)$ must be used if the rule (ae, in) is not used). This means that after a number $n \geq 0$ of steps, where only the rule $a \to aa$ is used (hence 2^{n+1} copies of a are produced), then we use the rule $a \to ee$.

If we have the same number of copies of a and of e, then all these objects are used by the rule (ae, in), and the computation stops – with only one communication step. The number generated is of the form 2^m, for some $m \geq 1$. If we have more copies of e than that of a, then the rule $(\#e, in)$ must be used. In this case, the computation never stops. If we have more copies of a than that of e, then either we use the rule $(\#e, in)$ or the rule (ae, in). Using rule $(\#e, in)$ will result to a computation that never ends. While all copies of e will be consumed by the application of the rule (ae, in). The remaining copies of a should evolve further. Thus, in a subsequent step we have to use again the communication rules. This means, the computation will have more than one communication steps, hence it is not accepted. Thus, (halting) computations with at most one communication step generate all and only the powers of 2.

Like before, a more systematic study of using the communication complexity as a tool to regulate computations remains to be done. We could ask several questions which are similar to those formulated in the previous section: Which is the power of finite communication? Does the communication thresholds induce an infinite hierarchy? What about other complexity measures, what about the mode EPC?

7 Communication Complexity of Solving Problems

In this section, let us consider our communication complexity measure as a standard complexity measure in solving problems. As we have already noticed, the measure $ComN$ is nothing else but the parameter time (number of computation steps) when we ignore the evolution steps and counting only the steps when at least one communication rule is used. This implies that the whole range of questions dealt within the theory of time complexity, [17], can be adopted for the new measure. Whether or not anything of interest can be obtained in this way remains to be explored. On the other hand, we could start with what it means to solve a problem Q using a class CL of P systems, and take "for free" the definition of complexity classes with a given communication effort.

In this context, we may start by having a problem Q, characterized by a size parameter *size* taking values from the natural numbers, and the set of instances I_Q taking Boolean values (*true* and *false*). We say that a family $\Pi(n), n \geq 1$, solves (uniformly) the problem Q if (i) each system $\Pi(n)$ is constructed starting from Q and n (hence not from the instances of size n of the problem), and (ii) introducing a code $cod(i_Q)$ of an instance i_Q of size n as a multiset in the skin region of system $\Pi(n)$, the computation halts if and only if i_Q is true (hence the family $\Pi(n), n \geq 1$, is sound and complete with respect to Q).

Note that we have said nothing here about the complexity of solving the problem, that is why we have said nothing about the complexity of constructing the systems $\Pi(n), n \geq 1$, starting from Q and n, neither on the complexity of computing $cod(i_Q)$ starting from i_Q, nor on the complexity of the halting computation in $\Pi(n)$. All these have natural definitions in the case of time complexity, especially when dealing with complexity classes at least polynomial: all these computations should be performed in at most polynomial time.

Here we have a problem: we want also to investigate complexity classes defined according to finite thresholds: such a class contains all problems which can be decided making use of a given number of communication steps, a natural number $k \geq 0$. The construction of the systems $\Pi(n), n \geq 1$, and the computation of $cod(i_Q)$ are done by a Turing machine, and for Turing machines we do not know what communication complexity means.

Therefore, we will ask, as usual questions in time complexity area, to have the systems $\Pi(n), n \geq 1$, constructed by a Turing machine in a polynomial time. The polynomial restriction, however, seems to be too permissive for the computation of the code of the problem instance: in a polynomial time, we can already solve the problem, hence $cod(i_Q)$ can be a single bit, 1 if the instance is true and 0 otherwise. This is not acceptable, and we do not have a general solution to this issue, that is why below we only consider problems whose true instances are described by numerical *relations*.

For instance, for a relation $rel_k \subseteq \mathbf{N}^k$, for some $k \geq 2$, we consider the problem $Qrel_k$ whose instances are of the form $i_{Qrel_k} = (n_1, n_2, \ldots, n_k)$, where $n_i \in \mathbf{N}$, $1 \leq i \leq k$, with

$$i_{Qrel_k}(n_1, \ldots, n_k) = \textit{ true} \text{ iff } (n_1, n_2, \ldots, n_k) \in rel_k.$$

In such a case, for an instance of size k we input in the skin region a multiset of the form $a_1^{n_1}a_2^{n_2}\dots a_k^{n_k}$, and we expect that the system $\Pi(k)$ halts if and only if the instance has the value *true*.

Another issue appears here, concerning the way the system is working: deterministically, non-deterministically, confluently. In the previous definition, the system is supposed non-deterministic, the instance of the problem is decided to be true if there is a computation which halts, irrespective of how many computations starting in the initial configuration do not halt. Of course, this can be changed, working only with deterministic systems.

In this setup, we say that the problem Q belongs to the complexity class $FComX(s), s \geq 0$, if each instance of size k of Q which is true leads to a halting computation δ of $\Pi(k)$ which has $ComX(\delta) \leq s$, for each $X \in \{N, R, W\}$ (as in Section 5). Note that in Section 5 we have considered families of sets of numbers which can be generated by EC P systems with the communication complexity bounded – more precisely, with *at least* one computation having the communication complexity bounded – while here we consider classes of problems associated with numerical relations, with the relation itself "recognized" by the system by means of computations with a bounded communication complexity. Of course, the classes $FComX(s)$ should also indicate the class of P systems used, but here we always consider EC P systems with quanta of energy, as in the previous sections (in particular, with minimal symport and antiport rules, moving only one object in a direction, with the help/consumption of a quantum of energy).

In what follows, we only briefly investigate the measure $ComN$, i.e., the number of steps when a communication rule is used.

According to the definitions, we have the inclusion $FComN(s) \subseteq FComN(s+1)$, for all $s \geq 0$, and it is expected that these inclusions are strict.

We can prove this for the first inclusion, making use of the problem associated with the equality relation,

$$eq_k = \{(n, n, \dots, n) \mid n \geq 0\} \subset \mathbf{N}^k,$$

for $k \geq 2$.

Actually, we believe that this problem can be used in order to prove that the hierarchy $FComN(s), s \geq 0$, is infinite. The constructive part is easy. We start by giving the system which decides Qeq_2, in order to make clear the general construction.

The system $\Pi(2)$ is given in Figure 4. The system contains only one object, d; the computation starts by introducing $a_1^{n_1}a_2^{n_2}$ in the skin region. Immediately, all objects a_1 are transformed into quanta of energy and all a_2 are primed. If $n_1 > n_2$, then at least one copy of e remains unused by the rule $(a_2'e, in)$, hence the rule (de, in) must be used and the computation never stops, because of the rule $d \to d$ in R_2. If $n_2 > n_1$, then, after consuming the n_1 copies of e, at least one a_2' remains to evolve forever by means of the rule $a_2' \to a_2'$ in R_1. If we have $n_1 = n_2$, then the computation can stop after two steps, with all copies of e consumed, all copies of a_2' introduced in membrane 2, and object d remaining idle in the skin region.

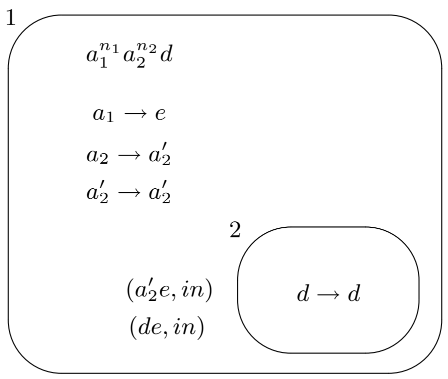


Fig. 4. An EC P system which decides Qeq_2

Thus, $Qeq_2 \in FComN(1)$. Note that the previous reasoning is valid for both modes CPE and CME, and below we will also work in these modes (with this semantics of our systems).

The fact that $Qeq_2 \notin FComN(0)$ is obvious: the evolution rules we use in our systems are non-cooperative, they cannot compare numbers, hence we need at least one communication step.

The relation $Qeq_k \in FComN(k-1)$ can be proved in general. The system $\Pi(k)$ used to this aim is given in Figure 5.

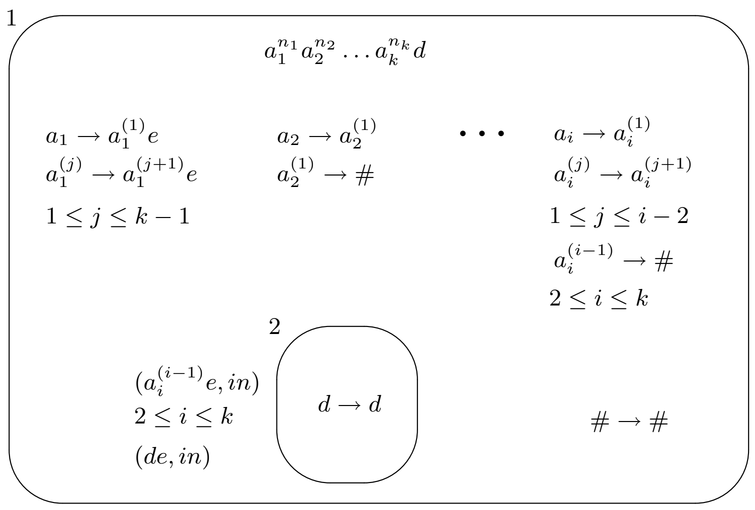


Fig. 5. An EC P system deciding Qeq_k

This system halts if and only if $n_1 = n_2 = \ldots = n_k$, and this is done by means of a computation which has $k-1$ communication steps. In a sequence, one checks whether or not $n_1 = n_2$, $n_1 = n_3$, and so on until $n_1 = n_k$, and each of these subproblems is solved in the way already shown in the case of $k = 2$ in Figure 4: rules associated with a_1 generate continuously copies of e; if their number is equal to the number of copies of $a_i^{(j)}$ produced at the same time, then the computation can continue without producing the trap object # and without introducing the object d in membrane 2 (in both cases, the computation would continue forever). This means that for each comparison we have a communication step, in total $k-1$, which shows that $Qeq_k \in FComN(k-1)$.

We *conjecture* that $Qeq_k \notin FComN(k-2)$, hence that the hierarchy $FComN(k)$, $k \geq 0$, is infinite.

A more systematic investigation of classes $FComN(k), k \geq 0$, remains to be carried out, maybe starting with further decision problems based on relations among numbers (hopefully, for one of them the strictness of the hierarchy will be obtained). Which problems can be decided in finite time? Which is the relation between usual complexity classes, with the time/space related by a function to the size of the input problem, and classes defined in a similar way for communication complexity measures?

What about considering other parameters than $ComN$, for instance, $ComR$? In some sense, counting the rules used in each communication step corresponds to the definition of so-called *Sevilla carpet*, see [2], [6]. Can this connection be made more precise?

These and many other questions remain to be answered. Then, all these problems – as well as those formulated in the previous sections – can be extended to other classes of P systems, as communication rules/tools appear in all of them.

8 Closing Remarks

Communication plays an essential role in P systems, so that it is rather natural to investigate this feature more carefully, in particular, to define parameters measuring the communication efforts of a computation, hence of a system. This question is also motivated in view of the fact that P systems are distributed parallel computing devices, hence a classic theory of communication complexity [7] is plausible in this framework. The present paper contributes only preliminarily to this research direction, by considering the effort of communication in EC P systems (in the form of quanta of energy consumed by the communication rules), and by proposing some dynamical communication complexity measures (and some problems and conjectures about them). In some sense, the main aim of this paper was to call the attention to this research area, almost untouched in membrane computing, and we hope that the reader will take this challenge and fill in this gap.

A direct continuation of the present paper is [14], where so-called dP systems are introduced and dP automata are briefly investigated; they are a natural framework for investigating communication complexity issues, thus proposing an answer to this question formulated in the present paper.

Acknowledgements

The work of H. Adorna was supported by the Engineering Research and Development for Technology (ERDT) Program of the Department of Science and Technology (DOST), the Philippines. The work of Gh. Păun and M.J. Pérez-Jiménez was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. M. Cavaliere: Evolution-communication P systems. *Membrane Computing. Proc. WMC 2002, Curtea de Argeş* (Gh. Păun et al., eds.), LNCS 2597, Springer, Berlin, 2003, 134–145.
2. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: Sevilla carpets associated with P systems. *Proc. Brainstorming Week on Membrane Computing* (M. Cavaliere et al., eds.), Tarragona Univ., TR 26/03, 2003, 135–140.
3. E. Csuhaj-Varjú: P automata. *Membrane Computing. Proc. WMC5, Milano, 2004* (G. Mauri et al., eds.), LNCS 3365, Springer, Berlin, 2005, 19–35.
4. E. Csuhaj-Varjú, M. Margenstern, G. Vaszil, S. Verlan: On small universal antiport P systems. *Theoretical Computer Sci.*, 372 (2007), 152–164.
5. J. Gruska: Descriptional complexity of context-free languages. *Proc. Symp. on Mathematical Foundations of Computer Science, MFCS*, High Tatras, 1973, 71–83.
6. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: Multidimensional Sevilla carpets associated with P systems. *Cellular Computing. Complexity Aspects* (M.A. Gutiérrez-Naranjo et al., eds.), Fenix Editora, Sevilla, 2005, 225–236.
7. J. Hromkovič: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing.* Springer, Berlin, 1997.
8. S.N. Krishna, A. Păun: Some universality results on evolution-communication P systems. *Proc. Brainstorming Week on Membrane Computing; Tarragona, February 2003* (M. Cavaliere et al., eds.), Technical Report 26/03, Rovira i Virgili University, Tarragona, 2003, 207–215.
9. A. Leporati, C. Zandron, G. Mauri: Simulating the Fredkin gate with energy-based P systems. *JUCS*, 10, 5 (2004), 600–619.
10. A. Leporati, G. Mauri, C. Zandron: Quantum sequential P systems with unit rules and energy assigned to membranes. *Membrane Computing, Proc. WMC6, Vienna, Austria, 2005* (R. Freund et al., eds.), LNCS 3850, Springer, Berlin, 2006, 310–325.
11. 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`).
12. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.

13. Gh. Păun: Further open problems in membrane computing, *Proc. Second Brainstorming Week on Membrane Computing, Sevilla, February 2004* (Gh. Păun et al., eds.), Technical Report 01/04 of Research Group on Natural Computing, Sevilla University, Spain, 2004, 354–365.
14. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed day in membrane computing: dP systems: *Int. J. of Computers, Communication and Control*, 5, 2 (2010), in press.
15. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.
16. Gh. Păun, Y. Suzuki, H. Tanaka: P systems with energy accounting. *Int. J. Computer Math.*, 78, 3 (2001), 343–364.
17. M.J. Pérez-Jiménez: A computational complexity theory in membrane computing. In *Pre-proceedings of WMC10, Curtea de Argeş, Romania, August 2009*, Technical Report 01/09 of Research Group on Natural Computing, Sevilla University, Spain, 2009, 82–105.
18. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Introducing a space complexity measure for P systems. *Intern. J. Computers, Communications and Control*, 4, 3 (2009), 301–310.
19. The P Systems Website: `http://ppage.psystems.eu`.

The Membrane Systems Language Class

Artiom Alhazov[1,2], Constantin Ciubotaru[1], Yurii Rogozhin[1], Sergiu Ivanov[1,3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
{artiom,chebotar,rogozhin,sivanov}@math.md
[2] IEC, Department of Information Engineering
Graduate School of Engineering, Hiroshima University
Higashi-Hiroshima 739-8527 Japan
[3] Technical University of Moldova, Faculty of Computers,
Informatics and Microelectronics,
Ştefan cel Mare 168, Chişinău MD-2004 Moldova

Summary. The aim of this paper is to introduce the class of languages generated by the transitional model of membrane systems without cooperation and without additional ingredients. The fundamental nature of these basic systems makes it possible to also define the corresponding class of languages it in terms of derivation trees of context-free grammars. We also compare this class to the well-known language classes and discuss its properties.

1 Introduction

Membrane computing is a theoretical framework of parallel distributed multiset processing. It has been introduced by Gheorghe Păun in 1998, and remains an active research area, see [6] for the comprehensive bibliography and [3],[4] for a systematic survey.

The configurations of membrane systems (with symbol objects) consist of multisets over a finite alphabet, distributed across a tree structure. Therefore, even such a simple structure as a word (i.e., a sequence of symbols) is not explicitly present in the system. To speak of languages as sets of words, one first needs to represent them in membrane systems, and there are a few ways to do it.

- Represent words by string objects. Rather many papers take this approach, see Chapter 7 of [4], but only few consider parallel operations on words. Moreover, a tuple of sets or multisets of words is already a quite complicated structure. The third drawback is that it is very difficult to define an elegant way of interactions between strings. Polarizations and splicing are examples of that; however, these are difficult to use in applications. In this paper we focus on symbol objects.

- Represent a word by a single symbol object, or by a few objects of the form (letter,position) as in, e.g., [1]. This only works for words of bounded length, i.e., one can speak about at most finite languages.
- Represent positions of the letters in a word by nested membranes. The corresponding letters can be then encoded by objects in the associated regions, membrane types or membrane labels. Working with such a representation, even implementing a rule $a \to bc$ requires sophisticated types of rules, like creating a membrane around existing membrane, as defined in [2].
- Consider letters as digits and then view words as numbers, or use some other encoding of words into numbers or multisets. Clearly, the concept of words ceases to be direct with such encoding. Moreover, implementing basic word operations in this way requires a lot of number processing, not to speak of parallel word operations.
- Do all the processing by multisets, and regard the order of sending the objects in the environment as their order in the output word. In case of ejecting multiple symbols in the same step, the output word is formed from any of their permutations. This paper is devoted to this way.

Informally, the class of languages we are interested in is the class generated by systems with parallel applications of non-cooperative rules that rewrite objects and/or send them between the regions. Surprisingly, this language class did not yet receive enough attention of researchers. Almost all known characterizations and even bounds for generative power of different variants of membrane systems with various ingredients and different descriptional complexity bounds are expressed in terms of REG, MAT, $ET0L$ and RE, their length sets and Parikh sets (and much less often in terms of FIN or other subregular classes, CF or CS). The membrane systems language class presents interest since we show it lies between regular and context-sensitive classes, being incomparable with well-studied intermediate ones.

2 Definitions

2.1 Formal language preliminaries

Consider a finite set V. The set of all words over V is denoted by V^*, the concatenation operation is denoted by $\bullet$ and the empty word is denoted by λ. Any set $L \subseteq V^*$ is called a language. For a word $w \in V^*$ and a symbol $a \in V$, the number of occurrences of a in w is written as $|w|_a$. The permutations of a word $w \in V^*$ are $\texttt{Perm}(w) = \{x \in V^* \mid |x|_a = |w|_a \forall a \in V\}$. We denote the set of all permutations of the words in L by $\texttt{Perm}(L)$, and we extend this notation to classes of languages. We use FIN, REG, LIN, CF, MAT, CS, RE to denote finite, regular, linear, context-free, matrix, context-sensitive and recursively enumerable families of languages, respectively. The family of languages generated by extended (tabled) interactionless L systems is denoted by $E(T)0L$. For more formal language preliminaries, we refer the reader to [5].

Throughout this paper we use string notation to denote the multisets. When speaking about membrane systems, keep in mind that the order in which symbols are written is irrelevant.

2.2 Transitional P systems

A membrane system is defined by a tuple

$\Pi = (O, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, i_0)$, where

O	is a finite set of objects,
μ	is a hierarchical structure of m membranes, bijectively labeled by $1, \cdots, m$, the interior of each membrane defines a region; the environment is referred to as region 0,
w_i	is the initial multiset in region $i, 1 \leq i \leq m$,
R_i	is the set of rules of region $i, 1 \leq i \leq m$,
i_0	is the output region; when languages are considered, $i_0 = 0$ is assumed.

The rules of a membrane systems have the form $u \to v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$ are written as a subscript, and target *here* is typically omitted. In case of non-cooperative rules, $u \in O$.

The rules are applied in maximally parallel way: no further rule should be applicable to the idle objects. In case of non-cooperative systems, the concept of maximal parallelism is the same as evolution in L systems: all objects evolve by the associated rules in the corresponding regions (except objects a in regions i such that R_i does not contain any rule $a \to u$, but these objects do not contribute to the result). The choice of rules is non-deterministic.

A sequence of transitions is called a computation. The computation halts when such a configuration is reached that no rules are applicable. The result of a (halting) computation is the *sequence* of objects sent to the environment (all the permutations of the symbols sent out in the same time are considered). The language $L(\Pi)$ generated by a P system Π is the union of the results of all computations. The class of languages generated by non-cooperative transitional P systems with at most m membranes is denoted by $LOP_m(ncoo, tar)$. If the number of membranes is not bounded, m is replaced by $*$ or omitted. If the target indications of the form in_j are not used, tar is replaced by out.

Example 1. To illustrate the concept of generating languages, consider the following P system:

$$\Pi = (\{a, b, c\}, [_1 \ ]_1, a^2, \{a \to \lambda, a \to a\ b_{out} c_{out}^2\}, 0).$$

Each of the two symbols a has a non-deterministic choice whether to be erased or to reproduce itself while sending a copy of b and two copies of c into the environment. Therefore, the contents of region 1 can remain a^2 for an arbitrary number $m \geq 0$ of steps, and after that at least one copy of a is erased. The other copy of a can reproduce itself for another $n \geq 0$ steps before being erased. Each of the first m steps, two copies of b and four copies of c are sent out, while in each of the next n steps, only one copy of b and two copies of c are ejected. Therefore, $L(\Pi) = (\texttt{Perm}(bccbcc))^*(\texttt{Perm}(bcc))^*$.

3 Context-free grammars and time-yield

Consider a non-terminal A in a grammar $G = (N, T, S, P)$. We denote by G_A the grammar (N, T, A, P) obtained by considering A as axiom in G.

A derivation tree in a context-free grammar is always a rooted tree with leaves labeled by terminals and all other nodes labeled by non-terminals. Rules of the form $A \to \lambda$ cause a problem, which can be solved by allowing to also label leaves by λ, or by transformation of the corresponding grammar. Note: throughout this paper by derivation trees we only mean finite ones. Consider a derivation tree τ.

The n-th level yield $\texttt{yield}_n$ of τ can be defined as follows:

We define $\texttt{yield}_0(\tau) = a$ if τ has a single node labeled by $a \in T$, and $\texttt{yield}_0(\tau) = \lambda$ otherwise.
Let k be the number of children nodes of the root of τ, and $\tau_1, \cdots, \tau_k$ be the subtrees of τ with these children as roots. We define $\texttt{yield}_{n+1}(\tau) = \texttt{yield}_n(\tau_1) \bullet \texttt{yield}_n(\tau_2) \bullet \cdots \bullet \texttt{yield}_n(\tau_k)$.

We now define the time yield L_t of a context-free grammar derivation tree τ, as the usual yield except the order of terminals is vertical from root instead of left-to-right, and the order of terminals at the same distance from root is arbitrary. We use $\prod$ to denote concatenation in the following formal definition:

$$L_t(\tau) = \prod_{n=0}^{\texttt{height}(\tau)} (\texttt{Perm}(\texttt{yield}_n(\tau))).$$

The time yield $L_t(G)$ of a grammar G is the union of time yields of all its derivation trees. The corresponding class of languages is

$$L_t(CF) = \{L_t(G) \mid G \text{ is a context-free grammar}\}.$$

Example 2. Consider a grammar $G_1 = (\{S, A, B, C\}, \{a, b, c\}, S, P)$, where

$$P = \{S \to SABC, S \to ABC, A \to A, B \to B, C \to C, A \to a, B \to b, C \to c\}.$$

We now show that $L_t(G_1) = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c > 0\} = L$. Indeed, all derivations of A are of the form $A \Rightarrow^* A \Rightarrow a$. Likewise, symbols B, C are also trivially rewritten an arbitrary number of times and then changes into a corresponding terminal. Hence, $L_t(G_{1A}) = \{a\}$, $L_t(G_{1B}) = \{b\}$, $L_t(G_{1C}) = \{c\}$.

For inclusion $L_t(G) \subseteq L$ it suffices to note that S always generates the same number of symbols A, B, C.

The converse inclusion follows from the following simulation: given a word $w \in L$, generate $|w|/3$ copies of A, B, C, and then apply their trivial rewriting in such way that the timing when the terminal symbols appear corresponds to their order in w.

Corollary 1. $L_t(CF) \not\subseteq CF$.

4 Membrane class via CF derivation trees

We first show that for every membrane system without cooperation, there is a system from the same class with one membrane, generating the same language.

Lemma 1. $LOP(ncoo, tar) = LOP_1(ncoo, out)$.

Proof. Consider an arbitrary transitional membrane system Π (without cooperation and without additional ingredients). The known technique of flattening the structure consists of transforming Π in the following way. Object a in region associated to membrane i is transformed into object (a, i) in the region associated to the single membrane. The alphabet, initial configuration and rules are transformed accordingly. Clearly, the configurations of the old system and the new system are isomorphic, and the output in the environment is the same.

Theorem 1. $L_t(CF) = LOP(ncoo, tar)$.

Proof. By Lemma 1, the statement is equivalent to $L_t(CF) = LOP_1(ncoo, out)$. Consider a P system $\Pi = (O, [_1\ \ ]_1, w, R, 0)$. We construct a context-free grammar $G = (O' \cup \{S\}, O, S, P \cup \{S \to w\})$, where S is a new symbol, $'$ is a morphism from O into new symbols and

$$\begin{aligned} P = \{a' \to u'v \mid (a \to u\ v_{out}) \in R,\ a \in O,\ u, v \in O^*\} \\ \cup\ \{a' \to \lambda \mid \neg\exists(a \to u\ v_{out}) \in R\}. \end{aligned}$$

Here v_{out} are those symbols on the right side of the rule in R which are sent out into the environment, and u are the remaining right-side symbols.

The computations of Π are identical to parallel derivations in G, except the following:

- Unlike G, Π does not keep track of the left-to-right order of symbols. This does not otherwise influence the derivation (since rules are context-free) or the result (since the order of non-terminals produced in the same step is arbitrary, and the timing is preserved).
- The initial configuration of Π is produced from the axiom of G in one additional step.

- The objects of Π that cannot evolve are erased in G, since they do not contribute to the result.

It follows that $L_t(CF) \supseteq LOP(ncoo, tar)$. To prove the converse inclusion, consider an arbitrary context-free grammar $G = (N, T, S, P)$. We construct a P system $\Pi = (N \cup T, [_1 \ ]_1, S, R, 0)$, where $R = \{a \to h(u) \mid (a \to u) \in R\}$, where h is a morphism defined by $h(a) = a$, $a \in N$ and $h(a) = a_{out}$, $a \in T$. The computations in Π correspond to parallel derivations in G, and the order of producing terminal symbols in G corresponds to the order of sending them to the environment by Π, hence the theorem statement holds.

We now present a few normal forms for the context-free grammars.

Lemma 2. *(First normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G'.*

- *the axiom does not appear in the right side of any rule, and*
- *if the left side is not the axiom, then the right side is not empty.*

Proof. The technique is essentially the same as removing λ-productions in classical theory of context-free grammars. Let $G = (N, T, S, P)$. First, introduce the new axiom S' and add a rule $S' \to S$. Compute the set $E \subseteq N$ of non-terminals that can derive λ by closure of

$$\begin{aligned}&(A \to \lambda) \longrightarrow (A \in E),\\ &(A \to A_1 \cdots A_k),\ (A_1, \cdots, A_k \in E) \longrightarrow (A \in E).\end{aligned}$$

Then replace productions $A \to u$ by $A \to h(u)$, where $h(a) = \{a, \lambda\}$ if $a \in E$ and $h(a) = a$ if $a \in N \cup T \setminus E$. Finally, remove λ-productions for all non-terminals except the axiom. Note that this transformation preserves not only the generated terminals, but also the order in which they are generated.

The First normal form shows that erasing can be limited to the axiom.

Lemma 3. *(Binary normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G'.*

- *the First normal form holds,*
- *the right side of any production is at most 2.*

Proof. The only concern in splitting the longer productions of $G = (N, T, S, P)$ in shorter ones is to preserve the order in which non-terminals are produced. The number

$$n = \lceil \log_2 \left(\max_{(A \to u) \in P} |u| \right) \rceil$$

is the number of steps sufficient to implement all productions of G by at most binary productions. Each production $p : A \to A_1 \cdots A_k$, $k \le 2^n$, is replaced by

$$\begin{aligned} A &\to p_{0,0}, \\ p_{i,j} &\to p_{i+1,2j}p_{i+1,2j+1} \text{ for } 0 \le i \le n-1, 0 \le j \le 2^i - 1, \\ p_{n,i-1} &\to A_i \text{ for } 1 \le i \le k, \\ p_{n,i} &\to \lambda \text{ for } k \le i \le 2^n. \end{aligned}$$

these productions implement a full binary tree of depth n, rooted in A with new symbols in intermediate nodes, and leaves labeled $A_1, \cdots, A_k$, all remaining leaves labeled λ (the first and last chain productions are given for the simplicity of the presentation). It only remains to convert the grammar obtained to the First normal form. Indeed, the derivations in the obtained grammar correspond to the derivation of the original one, with the slowdown factor of $n+2$, and the order of producing terminal symbols is preserved. Obviously, converting into the First normal form does not increase the size of the right side of productions.

The Binary normal form shows that productions with right side longer than two are not necessary.

Lemma 4. *(Third normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G'.*

- *the Binary normal form holds,*
- *$G' = (N, T, S, P')$ and every $A \in N$ is reachable,*
- *either $G' = (\{S\}, T, S, \{S \to S\})$, or $G' = (N, T, S, P')$ and for every $A \in N$, $L_t(G'_A) \neq \emptyset$.*

Proof. Consider a context-free grammar in the Binary normal form. First, compute the set $D \subseteq N$ of productive non-terminals as closure of

$$\begin{aligned} &(A \to u),\ (u \in T^*) \longrightarrow (A \in D) \\ &(A \to A_1 \cdots A_k),\ (A_1, \cdots, A_k \in D) \longrightarrow (A \in D). \end{aligned}$$

Remove all non-terminals that are not productive from N, and all productions containing them. If the axiom was also removed, then $L_t(G) = \emptyset$, hence we can take $G' = (\{S\}, T, S, \{S \to S\})$. Otherwise, compute the set $R \subseteq N$ of reachable non-terminals as closure of

$$\begin{aligned} &(S \in R), \\ &(A \in R),\ (A \to A_1 \cdots A_k) \longrightarrow (A_1, \cdots, A_k \in R). \end{aligned}$$

Remove all non-terminals that are not reachable from N, and all productions containing them. Note that all transformations preserve the generated terminals and the order in which they are produced, as well as the Binary normal form.

The Third normal form shows that never ending derivations are only needed to generate the empty language.

5 Comparison with known classes

Theorem 2. *$LOP(ncoo, tar) \supseteq REG$.*

Proof. Consider an arbitrary regular language. Then there exists a complete finite automaton $M = (Q, \Sigma, q_0, F, \delta)$ accepting it. We construct a context-free grammar $G = (Q, \Sigma, q_0, P)$, where $P = \delta \cup \{q \to \lambda \mid q \in F\}$. The order of symbols accepted by M corresponds to the order of symbols generated by G, and the derivation can only finish when the final state is reached. Hence, $L_t(G) = L(M)$, and the theorem statement follows.

Theorem 3. *$LOP(ncoo, tar) \subseteq CS$.*

Proof. Consider a context-free grammar $G = (N, T, S, P)$ in the First normal form. We construct a grammar $G' = (N \cup \{\#_1, L, R, F, \#_2\}, T, S', P')$, where

$$\begin{aligned} P' = \{&S' \to \#_1 LS\#_2, L\#_2 \to R\#_2, \#_1 R \to \#_1 L, \#_1 R \to F, F\#_2 \to \lambda\} \\ &\cup \{LA \to uL \mid (A \to u) \in P\} \cup \{La \to aL, Fa \to aF \mid a \in T\} \\ &\cup \{aR \to Ra \mid a \in N \cup T\}. \end{aligned}$$

The symbols $\#_1, \#_2$ mark the edges, the role of symbol L is to apply productions P to all non-terminals, left-to-right, while skipping the terminals. While reaching the end marker, symbol L changes into R and returns to the beginning marker, where it either changes back to L to iterate the process, or to F to check whether the derivation is finished.

Hence, $L(G') = L_t(G)$. Note that the length of sentential forms in any derivation (of some word with n symbols in G') is at most $n + 3$, because the only shortening productions are the ones removing $\#_1, \#_2$ and F, and each should be applied just once. Therefore, $L_t(G) \in CS$, and the theorem is proved.

We now proceed to showing that the membrane systems language class does not contain the class of linear languages. To show this, we first define the notions of unbounded yield and unbounded time of a non-terminal.

Definition 1. *Consider a grammar $G = (N, T, S, P)$. We say that $A \in N$ has an unbounded yield if $L_t(G_A)$ is an infinite language, i.e., there is no upper bound on the length of words generated from A.*

It is easy to see that $L_t(G_A)$ is infinite if and only if $L(G_A)$ is infinite; decidability of this property is well-known from the theory of context-free grammars.

Definition 2. *Consider a grammar $G = (N, T, S, P)$. We say that $A \in N$ has unbounded time if the set of all derivation trees (for terminated derivations) in G_A is infinite, i.e., there is no upper bound on the number of parallel steps of terminated derivations in G_A.*

It is easy to see that A has unbounded time if $L(G_A) \neq \emptyset$ and $A \Rightarrow^+ A$, so decidability of this property is well-known from the theory of context-free grammars.

Lemma 5. *Let $G = (N, T, P, S)$ be a context-free grammar in the Third normal form. If for every rule $(A \to BC) \in P$, symbol B does not have unbounded time, than $L_t(G) \in REG$.*

Proof. Assume the premise of the lemma holds. Let F be the set of the first symbols in the right sides of all binary productions. Then there exists a maximum m of time bounds for the symbols in F. For every such symbol $B \in F$ there also exists a finite set $t(B)$ of derivation trees in G_B. Let $t = \{\emptyset\} \cup \bigcup_{B \in F} t(B)$ be the set of all such derivation trees, also including the empty tree. We recall that t is finite.

We perform the following transformation of the grammar: we introduce non-terminals of the form $A[\tau_1, \cdots, \tau_{m-1}]$, $A \in N \cup \emptyset$, $\tau_i \in t$, $1 \le i \le m-1$. The new axiom is $S[\emptyset, \cdots, \emptyset]$. Every binary production $A \to BC$ is replaced by productions

$$A[\tau_1, \cdots, \tau_{m-1}] \to \mathtt{yield}_0(\tau)\mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) \\ C[\tau, \tau_1, \cdots, \tau_{m-2}] \text{ for all } \tau \in t(B).$$

Accordingly, productions $A \to C$, $C \in N$ are replaced by productions

$$A[\tau_1, \cdots, \tau_{m-1}] \to \mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) C[\emptyset, \tau_1, \cdots, \tau_{m-2}],$$

and productions $A \to a$, $a \in T$ are replaced by productions

$$A[\tau_1, \cdots, \tau_{m-1}] \to a\ \mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) \emptyset[\emptyset, \tau_1, \cdots, \tau_{m-2}].$$

Finally, $\emptyset[\emptyset, \cdots, \emptyset] \to \lambda$ and

$$\emptyset[\tau_1, \cdots, \tau_{m-1}] \to \mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) \emptyset[\emptyset, \tau_1, \cdots, \tau_{m-2}].$$

In simple words, if the effect of one symbol is limited to m steps, then the choice of the corresponding derivation tree is memorized as an index in the other symbol, and needed terminals are produced in the right time. In total, m indexes suffice. It is easy to see that underlying grammar is regular, since only one non-terminal symbol is present.

Lemma 6. *$L = \{a^n b^n \mid n \ge 1\} \notin LOP(ncoo, tar)$.*

Proof. Suppose there exists a context-free grammar $G = (N, T, S, P)$ in the Third normal form such that $L_t(G) = L$. Clearly, there must be a rule $A \to BC$ or $A \to CB \in P$ such that both B and C have unbounded time (by Lemma 5, since $L \notin REG$) and C has unbounded yield (since $L \notin FIN$).

Clearly, languages generated by any non-terminal from N must be scattered subwords of words from L, otherwise G would generate some language not in L. Thus, $L_t(G_B), L_t(G_C) \subseteq \{a^i b^j \mid i, j \ge 0\}$. It is not difficult to see that G_C must produce both symbols a and b. Indeed, since the language generated from C is infinite, substituting derivation trees for C with different numbers of one letter must preserve the balance of two letters. We now consider two cases, depending on whether $L_t(G_B) \subseteq a^*$.

If B only produces symbols a, then consider the shortest derivation tree τ in G_C. Since B has unbounded time, some symbol a can be generated after the first letter b appears in τ, so $L_t(G)$ generates some word not in L, which is a contradiction.

Now consider the case when B can produce a symbol b in some derivation tree τ in G_B. On one hand, a bounded number of letters a can be generated from B and C before the first letter b appears in τ; on the other hand, C has unbounded yield. Therefore, varying derivations under C we obtain a subset of $L_t(G)$ which is infinite, but the number of leading symbols a is bounded, so $L_t(G)$ contains words not in L, which is a contradiction.

Corollary 2. *$LIN \not\subseteq LOP(ncoo, tar)$.*

Lemma 7. *The class $LOP(ncoo, tar)$ is closed under permutations.*

Proof. For a given grammar $G = (N, T, S, P)$, consider a transformation where the terminal symbols a are replaced by non-terminals a_N throughout the description of G, and then the rules $a_N \to a_N$, $a_N \to a$, $a \in T$ are added to P. In a way similar to the first example, the order in which terminals are generated is arbitrary.

Corollary 3. *$\texttt{Perm}(REG) \subseteq LOP(ncoo, tar)$.*

Proof. Follows from regularity theorem 2 and permutation closure lemma 7.

The results of comparison of the membrane system class with the well-known language classes can be summarized as follows:

Theorem 4. *$LOP(ncoo, tar)$ strictly contains REG and $\texttt{Perm}(REG)$, is strictly contained in CS, and is incomparable with LIN and CF.*

Proof. All inclusions and incomparabilities have been shown in or directly follow from Theorem 2, Corollary 3, Theorem 3, Corollary 2 and Corollary 1 with Theorem 1. The strictness of the first inclusions follows from the fact that REG and $\texttt{Perm}(REG)$ are incomparable, while the strictness of the latter inclusion holds since $LOP(ncoo, tar)$ only contains semilinear languages.

The lower bound can be strengthened as follows:

Theorem 5. *$LOP(ncoo, tar) \supseteq REG \bullet \texttt{Perm}(REG)$.*

Proof. Indeed, consider the construction from the regularity theorem. Instead of erasing the symbol corresponding to the final state, rewrite it into the axiom of the grammar generating the second regular language, to which the permutation technique is applied.

Example 3. $LOP(ncoo, tar) \ni L_2 = \bigcup_{m,n \geq 1} (abc)^m \texttt{Perm}((def)^n)$.

6 Closure properties

It has been shown above that the class of languages generated by basic membrane systems is closed under permutations. We now present a few other closure properties.

Lemma 8. *The class $LOP(ncoo, tar)$ is closed under erasing/renaming morphisms.*

Proof. Without restricting generality, we assume that the domain and range of a morphism h are disjoint. For a given grammar $G = (N, T, S, P)$, consider a transformation where the terminal symbols become non-terminals and the rules $a \to h(a)$, $a \in T$ are added to P. It is easy to see that the new grammar generates exactly $h(L_t(G))$.

Corollary 4. $\{a^n b^n c^n \mid n \geq 1\} \notin LOP(ncoo, tar)$.

Proof. Assuming the contrary and applying morphism defined by $h(a) = a'$, $h(b) = b'$, $h(c) = \lambda$, and then a morphism removing primes, we obtain a contradiction with $L = \{a^n b^n \mid n \geq 1\} \notin LOP(ncoo, tar)$ from Lemma 6.

Corollary 5. *$LOP(ncoo, tar)$ is not closed under intersection with regular languages.*

Proof. By Example 2, $L = \{w \in T^* \mid |w|_a = |w|_b = |w|_c > 0\}$ belongs to the membrane systems language class. However, $L \cap a^*b^*c^* = \{a^n b^n c^n \mid n \geq 1\}$ does not, by Corollary 4.

Theorem 6. *$LOP(ncoo, tar)$ is closed under union and not closed under intersection or complement.*

Proof. The closure under union follows from adding a new axiom and productions of non-deterministic choice between multiple axioms. The class is not closed under intersection because it contains all regular languages (Theorem 2) and is not closed under intersection with them (Corollary 5). It follows that this class is not closed under complement, since intersection is the complement of union of complements.

Lemma 9. $L = \bigcup_{m,n \geq 1} \mathtt{Perm}((ab)^m)c^n \notin LOP(ncoo, tar)$.

Proof. Suppose there exists a context-free grammar $G = (N, T, S, P)$ in the Third normal form such that $L_t(G) = L$. Clearly, there must be a rule $A \to BC$ or $A \to CB \in P$ such that both B and C have unbounded time (by Lemma 5, since $L \notin REG$) and C has unbounded yield (since $L \notin FIN$). By choosing as $A \to BC$ or $A \to CB$ the rule satisfying above requirements which is first applied in some derivation of G, we make sure that all three letters a, b, c appear in words of $L_t(G_A)$.

Clearly, languages generated by any non-terminal from N must be scattered subwords of words from L, otherwise G would generate some language not in

L. Thus, $L_t(G_B), L_t(G_C) \subseteq \{a,b\}^*c^*$. We now consider two cases, depending on whether $L_t(G_B) \subseteq \{a,b\}^*$.

If B only produces symbols a, b, then consider the shortest derivation tree τ in G_C. Since B has unbounded time, some symbol a or b can be generated after the first letter c appears in τ, so $L_t(G)$ generates some word not in L, which is a contradiction.

Now consider the case when B can produce a symbol c in some derivation tree τ in G_B. On one hand, a bounded number of letters a, b can be generated from B and C before the first letter c appears in τ; on the other hand, C has unbounded yield. Therefore, varying derivations under C we obtain a subset of $L_t(G)$ which is infinite, but the number of leading symbols a, b is bounded, so $L_t(G)$ contains words not in L, which is a contradiction.

Corollary 6. *$LOP(ncoo, tar)$ is not closed under concatenation or taking the mirror image.*

Proof. Since $\bigcup_{m\geq 1} \mathtt{Perm}((ab)^m) \in \mathtt{Perm}(REG) \subseteq LOP(ncoo, tar)$ by Corollary 3 and $c^+ \in REG \subseteq LOP(ncoo, tar)$ by Theorem 2, the first part of the statement follows from Lemma 9. Since $\bigcup_{m,n\geq 1} c^n\mathtt{Perm}((ab)^m) \in REG \bullet \mathtt{Perm}(REG) \subseteq LOP(ncoo, tar)$ by Theorem 5, the second part of the statement also follows from Lemma 9.

7 Conclusions

In this paper we have reconsidered the class of languages generated by transitional P systems without cooperation and without additional control. It was shown that one membrane is enough, and a characterization of this class was given via derivation trees of context-free grammars. Next, three normal forms were given for the corresponding grammars. It was than shown that the membrane systems language class lies between regular and context-sensitive classes of languages, and it is incomparable with linear and with context-free languages. Then, the lower bound was strengthened to $REG \bullet \mathtt{Perm}(REG)$.

The membrane systems language class was shown to be closed under union, permutations, erasing/renaming morphisms. It is not closed under intersection, intersection with regular languages, complement, concatenation or taking the mirror image.

The following are examples of questions that are still not answered.

- Clearly, $LOP(ncoo, tar) \not\supseteq MAT$. What about $LOP(ncoo, tar) \subseteq MAT$?
- Is $LOP(ncoo, tar)$ closed under arbitrary morphisms? The difficulty is to handle $h(a) = bc$ if many symbols a can be produced in the same step.
- Look for sharper lower and upper bounds.

Acknowledgments

Artiom Alhazov gratefully acknowleges the support of the Japan Society for the Promotion of Science and the Grant-in-Aid for Scientific Research, project 20·08364. All authors acknowledge the support by the Science and Technology Center in Ukraine, project 4032.

References

1. A. Alhazov, D. Sburlan: Static Sorting P Systems. *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Natural Computing Series, Springer-Verlag, 2005, 215–252.
2. F. Bernardini, M. Gheorghe: Language Generating by means of P Systems with Active Membranes. *Brainstorming Week on Membrane Computing*, Technical Report 26, Rovira i Virgili University, Tarragona, 2003, 46–60.
3. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
4. Gh. Păun, G. Rozenberg, A. Salomaa, Eds.: *Handbook of Membrane Computing.* Oxford University Press, 2009.
5. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, vol. 1-3, Springer, 1997.
6. P systems webpage. `http://ppage.psystems.eu/`.

Array Tissue-like P Systems

Hepzibah A. Christinal[1], Daniel Díaz-Pernil[1],
Miguel A. Gutiérrez-Naranjo[2], Mario J. Pérez-Jiménez[2]

[1] Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics I
{hepzi,sbdani}@us.es

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
{magutier,marper}@us.es

University of Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain

Summary. Array grammars have been studied in the framework of Membrane Computing by using rewriting rules from transition P systems. In this paper we present a new approach to dealing with array grammars by using tissue-like P systems and present an application to the segmentation of images in two dimensional computer graphics.

1 Introduction

Array grammars can be considered as a straightforward extension of string grammars to two dimensional pictures. Such pictures are sets of symbols placed in the points with integer coordinates of the plane. They have been widely studied and have a large tradition in the literature (see, e.g. [2, 6, 16, 22]).

Recently, Membrane Computing has also approximated to array grammars by setting bridges between both areas (see, e.g. [1, 14, 20]). The basic idea in such approaches is considering an array (i.e., a finite set of objects placed in points of the plane with integer coordinates) as a P system object and using rewriting rules of the type used in transition P systems [13] for replacing it. The type of rule used is $x \rightarrow y(tar)$ where $x \rightarrow y$ is a context-free rule and $tar \in here, out, in$ is the target which indicates the membrane where the generated object will be placed. Such rewriting rules capture the idea of *array production* $p : \mathcal{A} \rightarrow \mathcal{B}$ with $\mathcal{A}$ and $\mathcal{B}$ arrays.

In this paper we present a new approach for linking Membrane Computing to array grammars. Instead of using transition P systems to handle the arrays we propose to use tissue-like P systems. This approach allows us to use the power of symport-antiport rules for designing Membrane Computing algorithms which deal with array objects. In such P system model, the rules are of type $(i, u/v, j)$ with the following interpretation: If the multiset u occurs in a membrane with label i and the multiset v occurs in a membrane with label j, both multiset can be interchanged. We consider an extension of this type of rules. We will consider that two arrays A and B can appear (respectively) in the multisets u and v. The semantics of such rule will be explained below, but the intuition is that the arrays in the membranes i and j will be partially modified.

As a case study, we present an application of array tissue-like P systems to the *Segmentation Problem* in computer vision.

Segmentation in computer vision (see [8]), refers to the process of partitioning a digital image into multiple segments (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze for an human. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

In the literature, there exists different techniques to segment an image. Some of them are clustering methods [23], histogram-based methods [21], Watershed transformation methods [25] or graph partitioning methods [24]. Some of the practical applications of image segmentation are medical imaging [23], the study of anatomical structure, locate objects in satellite images (roads, forests, etc.) [19] or face recognition [7] among others.

The paper is organized as follows: First we briefly recall some basic definitions related to graphs and multisets and introduce our definition of pixel and array. Next, we introduce a new P system model called *Array tissue-like P systems* on the basis of *tissue P systems*. In Section 4, this P system model is used to find a solution to the segmentation problem in Digital Image.

2 Definitions

An *alphabet*, Σ, is a non-empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (of length 0) is denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$. A *language* over Σ is a subset of Σ^*. A *multiset* over a set A is a pair (A, f) where $f:\ A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its support is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite). If $m = (A, f)$ is a finite multiset over A, then it is denoted by $m = a_1^{f(a_1)} a_2^{f(a_2)} \cdots a_k^{f(a_k)}$, where $supp(m) = \{a_1, \ldots, a_k\}$, and for each element a_i, $f(a_i)$ is called the multiplicity of a_i. An *undirected graph* G is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges, each one of which is an (unordered) pair of (different) vertices. If $\{u, v\} \in E$, we say that u is *adjacent* to v (and also v is *adjacent* to u). The *degree* of $v \in V$ is the number of adjacent vertices to v. In what follows we assume that the reader is already familiar with the basic notions and the terminology underlying P systems[3].

Next, we give a formalization of the arrays considered in this paper.

Definition 1. *Given a finite set V, called an* alphabet of colors*, a* pixel *on V is a pair $\langle \mathbf{x}, v\rangle$ such that $\mathbf{x} \in \mathbb{Z}^2$ and $v \in V$. An* array *on V, A, is a finite set of pixels such that if $\langle \mathbf{x}_1, v_1\rangle, \langle \mathbf{x}_2, v_2\rangle \in A$ and $v_1 \neq v_2$ then $\mathbf{x}_1 \neq \mathbf{x}_2$. Finally, the* support *of the array A is the set $supp(A) = \{\mathbf{x} \in \mathbb{Z}^2 \mid \exists v \in V$ such that $\langle \mathbf{x}, v\rangle \in A\}$.*

Given an array A and $\mathbf{z} \in \mathbb{Z}^2$, we will denote by $A + \mathbf{z}$ the set

$$A + \mathbf{z} = \{\langle \mathbf{x} + \mathbf{z}, v\rangle \mid \langle \mathbf{x}, v\rangle \in A\}$$

Example 1. Let $V = \{R, G, B\}$ be the alphabet of colors and A the array on V $A = \{\langle(3, 2), R\rangle, \langle(3, 3), G\rangle, \langle(5, 5), G\rangle\}$. Let us consider $\mathbf{z} = (-2, 1) \in \mathbb{Z}^2$. The array $A + \mathbf{z}$ is $\{\langle(1, 3), R\rangle, \langle(1, 4), G\rangle, \langle(3, 6), G\rangle\}$.

If there are no confusion about the alphabet of colors, we will omit it and we talk about *pixels*. As usual, we will denote by V^{*2} the set of all two dimensional arrays over V.

3 Array Tissue-like P Systems

In the initial definition of the cell-like model of P systems [12], membranes are hierarchically arranged in a tree-like structure. Its biological inspiration comes from the morphology of cells, where small vesicles are surrounded by larger ones. This biological structure can be abstracted into a tree-like graph, where the root represents the skin of the cell (i.e. the outermost membrane) and the leaves represent membranes that do not contain any other membrane (elementary membranes). Besides, two nodes in the graph are connected if they represent two membranes such that one of them contains the other one.

In *tissue P systems*, the tree-like membrane structure is replaced by a general graph. This model has two biological inspirations (see [9, 10]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules, which were introduced as communication rules for P systems in [11]. 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 $q \geq 1$ with input is a tuple of the form

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

where

[3] We refer to [13] for basic information in this ares, to [15] for a comprehensive presentation and the web site [26] for the up-to-date information.

1. Γ is a finite *alphabet*, whose symbols will be called *objects*,
2. $\Sigma(\subset \Gamma)$ is the input alphabet,
3. $\mathcal{E} \subseteq \Gamma$ (the objects in the environment),
4. $w_1, \dots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration,
5. $\mathcal{R}$ is a finite set of communication rules of the following form: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \dots, q\}, i \neq j,\ u, v \in \Gamma^*$,
6. $i_\Pi \in \{0, 1, 2, \dots, q\}$,
7. $o_\Pi \in \{0, 1, 2, \dots, q\}$.

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

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

The communication rule $(i, u/v, j)$ can be applied over two cells labeled by i and j such that u is contained in cell i and v is contained in cell j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells. Note that if either $i = 0$ or $j = 0$ then the objects are interchanged between a cell and the environment.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered). In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we apply a maximal set of rules.

In order to understand how we can obtain a computation of one of these P systems we present an example of them:

Consider us the following tissue-like P system

$$\Pi' = (\Gamma, \Sigma, \mathcal{E}, w_1, w_2, \mathcal{R}, i_\Pi, o_\Pi)$$

where

1. $\Gamma = \{a, b, c, d, e\}$,
2. $\Sigma = \emptyset$,
3. $\mathcal{E} = \{a, b, e\}$,
4. $w_1 = a^3\, e$, $w_2 = b^2\, c\, d$,
5. $\mathcal{R}$ is the following set of communication rules
   (a) $(1, a/b, 2)$,
   (b) $(2, c/b^2, 0)$,
   (c) $(2, d/e^2, 0)$,
   (d) $(1, e/\lambda, 0)$,
6. $i_\Pi = 1$,
7. $o_\Pi = 0$

We can observe the initial configuration of this system in the Figure 1 (a). We have four rules to apply. First rule is $(1, a/b, 2)$. The rule can be applied whenever an object 'a' is founded in cell 1 and one copy of 'b' appear in cell 2. This rule sends 'a' to cell 2 and 'b' from cell 2 to cell 1. Rule 2 is $(2, c/b^2, 0)$ and implies that when symbol 'c' present in cell 2 then this rule takes two copies of 'b' from environment and sends 'c' to the environment (i.e. cell 0). Rule 3 is similar to rule 2. Rule 4, $(1, e/\lambda, 0)$, sends the object 'e' to the environment. So, as we have 3 copies of 'a' and 1 copy of 'e' in cell 1 and 2 copies of 'b', one copy of 'c' and two copies of 'd' appear in cell 2. Then, all the rules can be applied in a parallel manner. Figure 1(b) show the next configuration of the system after applying the rules. If reader observes the initial elements in the environment of a tissue-like P systems (in this case a, b), one can observe the number of the copies of these elements always appear as one, because we have an arbitrary large amount of copies of them. The only objects changing its number of copies in the environment during a computation are the elements were not appear there initially. In this example, d has two copies because it is not an initial element of the environment.

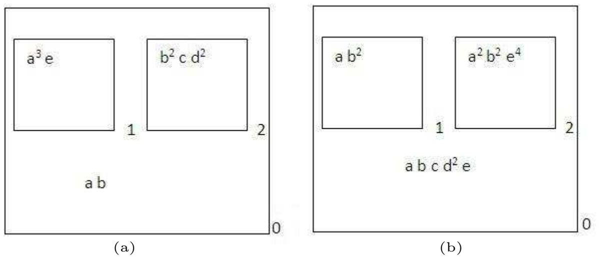


Fig. 1. (a) Initial Configuration of system Π' (b) Following Configuration of Π'

Next, we introduce a modification of this model in order to deal with arrays. An *array tissue-like P system* of degree $q \geq 1$ with input is a tuple of the form

$$\Pi = (\Gamma, V, \mathcal{E}, w_0, w_1, \ldots, w_q, A_1, \ldots, A_q, \mathcal{R}, i_\Pi, o_\Pi),$$

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*,
2. V is the *alphabet of colors* verifying $V \cap \Gamma = \emptyset$.
3. $\mathcal{E}$ is a finite subset of arrays on V.
4. $w_0, w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration,
5. $A_1, \ldots, A_n$ are arrays on V, placed on the corresponding cells at the initial configuration.
6. $\mathcal{R}$ is a finite set of communication rules of the following form: $(i, u_i W_i / u_j W_j, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u_i, u_j \in \Gamma^*$ and W_i, W_j two arrays on V.

7. $i_\Pi \in \{0, 1, 2, \ldots, q\}$ is the input cell.
8. $o_\Pi \in \{0, 1, 2, \ldots, q\}$ is the output cell.

In a similar way to tissue-like P systems, an *array tissue-like P system* of degree $q \geq 1$ can be seen as a set of q cells (each one consisting of an elementary membrane) labeled by $1, 2, \ldots, q$. We will use 0 to refer to the label of the environment, i_Π and o_Π denote the input region and the output region (which can be the region inside a cell or the environment) respectively.

The strings $w_1, \ldots, w_q$ describe the multisets of objects placed in the q cells of the system. We interpret that w_0 is the set of objects placed in the environment, each one of them available in an arbitrary large amount of copies.

For each $i \in \{1, \ldots, q\}$, each A_i is an array placed in the cell i in the initial configuration and $\mathcal{E}$ is the set of arrays placed in the environment, each one of them available in an arbitrary large amount of copies. The empty array $\emptyset$ always belongs to $\mathcal{E}$. For all the non-empty copies, we will consider that the leftmost pixel of the bottom row in the array corresponds to the coordinates $(0, 0)$.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered), regardless if the environment is involved or not. In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we apply a maximal set of rules.

The main difference with respect tissue-like P systems is related to the application of the rules.

Definition 2. *Let us consider two index i, j such that $i \neq 0 \neq j$ and two non-empty arrays W_i and W_j. The communication rule $(i, u_i W_i / u_j W_j, j)$ is* applicable *over two cells labeled by i and j if the following conditions are verified:*

- *u_i is contained in cell i and u_j is contained in cell j*
- *There exist two arrays, A_i in the cell i and A_j in the cell j and two pairs $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{Z}^2$ such that*
  (a) $W_i + \mathbf{z}_1 \subseteq A_i$
  (b) $W_j + \mathbf{z}_2 \subseteq A_j$
  (c) $supp(W_i) \cap supp(W_j) \neq \emptyset$
  (d) $supp(A_i - (W_i + \mathbf{z}_1)) \cap supp(W_j + \mathbf{z}_1) = \emptyset$
  (e) $supp(A_j - (W_j + \mathbf{z}_2)) \cap supp(W_i + \mathbf{z}_2) = \emptyset$

The application of this rule means that the objects of the multisets represented by u_i and u_j are interchanged between the two cells. The arrays, A_i in the cell i and A_j in the cell j are substituted by A'_i and A'_j respectively, where

$$A'_i = (A_i - (W_i + \mathbf{z}_1)) \cup (W_j + \mathbf{z}_1) \qquad A'_j = (A_j - (W_j + \mathbf{z}_2)) \cup (W_i + \mathbf{z}_2)$$

Note that if either A_i or A_j is the empty array, then the rule is not applicable.

Example 2. Let us suppose that we have two cells with labels 1 and 2 with the following objects and arrays, $[\,z_2c_3A_1\,]_1$ and $[\,d_3k_3bA_2\,]_2$, with z_2, c_3, d_3, k_3, b objects and A_1, A_2 arrays over $\{R, B, G\}$

$$\begin{aligned} A_1 &= \{\langle(1,1),G\rangle, \langle(1,2),G\rangle, \langle(2,2),R\rangle, \langle(2,3),B\rangle\} \\ A_2 &= \{\langle(5,5),G\rangle, \langle(6,5),G\rangle, \langle(6,6),G\rangle\} \end{aligned}$$

Let us consider the rule $r_1 \equiv (1,\, z_2W_1\,/\,d_3k_3W_2,\, 2)$ where W_1 and W_2 are the arrays

$$\begin{aligned} W_1 &= \{\langle(7,0),G\rangle, \langle(7,1),G\rangle, \langle(8,1),R\rangle\} \\ W_2 &= \{\langle(7,1),G\rangle, \langle(8,1),G\rangle\} \end{aligned}$$

We will check that r_1 is applicable to the cells 1 and 2

- z_2 is contained in the cell 1 and d_3k_3 is contained in the cell 2.
- Let us consider $\mathbf{z}_1 = (-6,1) \in \mathbb{Z}^2$ and $\mathbf{z}_2 = (-2,4) \in \mathbb{Z}^2$
  (a) $W_1 + \mathbf{z}_1 = \{\langle(1,1),G\rangle, \langle(1,2),G\rangle, \langle(2,2),R\rangle\} \subseteq A_1$
  (b) $W_2 + \mathbf{z}_2 = \{\langle(5,5),G\rangle, \langle(6,5),G\rangle\} \subseteq A_2$
  (c) $supp(W_i) \cap supp(W_j) = \{((7,0),(7,1),(8,1)\} \cap \{(7,1),(8,1)\} \neq \emptyset$
  (d) $A_1 - (W_1 + \mathbf{z_1}) = \{\langle(2,3),B\rangle\}$ and $W_2 + \mathbf{z}_1 = \{\langle(1,2),G\rangle, \langle(2,2),G\rangle\}$. By considering their supports we have $supp(A_1 - (W_1 + \mathbf{z_1})) = \{(2,3)\}$ and $supp(W_2 + \mathbf{z}_1) = \{(1,2),(2,2)\}$, then

$$supp(A_1 - (W_1 + \mathbf{z_1})) \cap supp(W_2 + \mathbf{z}_1) = \emptyset$$

  (e) $A_2 - (W_2 + \mathbf{z_2}) = \{\langle(6,6),G\rangle\}$ and $W_1 + \mathbf{z}_2 = \{\langle(5,4),G\rangle, \langle(5,5),G\rangle, \langle(6,5),R\rangle\}$. By considering their supports we have $supp(A_2 - (W_2 + \mathbf{z_2})) = \{(6,6)\}$ and $supp(W_1 + \mathbf{z}_2) = \{(6,4),(5,5),(6,5)\}$, then

$$supp(A_2 - (W_2 + \mathbf{z_2})) \cap supp(W_1 + \mathbf{z}_2) = \emptyset$$

The rule r_1 is applicable to the cells 1 and 2, and the result of applying the rule is $[\,d_3k_3c_3A_1'\,]_1$ and $[\,z_2bA_2'\,]_2$ where

$$\begin{aligned} A_1' &= (A_1 \,-\, (W_1 + \mathbf{z}_1)) \cup (W_2 + \mathbf{z}_1) \\ &= \{\langle(2,3),B\rangle, \langle(1,2),G\rangle, \langle(2,2),G\rangle\} \\ A_2' &= (A_2 \,-\, (W_2 + \mathbf{z}_2)) \cup (W_1 + \mathbf{z}_2) \\ &= \{\langle(6,6),G\rangle, \langle(5,4),G\rangle, \langle(5,5),G\rangle, \langle(6,5),R\rangle\} \end{aligned}$$

Next, we define the applicability of a rule if one of the regions involved is the environment and the arrays are not empty.

Definition 3. *Let us consider an index $i \neq 0$ and two non-empty arrays W_i and W_0. The communication rule $(i, u_iW_i/u_0W_0, 0)$ is* applicable *over two cells labeled by i and 0 if the following conditions are verified:*

- *u_i is contained in cell i and u_0 is contained in cell 0*
- *There exist an array A_i in the cell i and two pairs $\mathbf{z}_i, \mathbf{z}_0 \in \mathbb{Z}^2$ such that*

(a) $W_i + \mathbf{z}_i \subseteq A_i$
(b) $supp(W_i + \mathbf{z}_i) \cap supp(W_0 + \mathbf{z}_0) \neq \emptyset$
(c) $supp(A_i \; - \; (W_i + \mathbf{z}_i)) \cap supp(W_0 + \mathbf{z}_0) = \emptyset$

The application of this rule means that the objects of the multisets represented by u_i is removed from the cell i and substituted by the multiset represented by u_0. The arrays, A_i in the cell i is substituted by A'_i where

$$A'_i = (A_i \; - \; (W_i + \mathbf{z}_i)) \cup (W_0 + \mathbf{z}_0)$$

Example 3. Let us suppose the cell 1 with the following objects and arrays, $[\, z_2^3 c_3 A_1 \,]_1$ and A_1 the array over $\{R, B, G\}$

$$\begin{aligned} A_1 = \{ \; & \langle (5,2), R\rangle, \langle (6,2), B\rangle, \langle (7,2), G\rangle, \langle (8,2), B\rangle\} \\ & \langle (9,2), R\rangle, \langle (6,1), B\rangle, \langle (8,1), B\rangle\} \end{aligned}$$

Let us consider the rule $r_1 \equiv (1, \, z_2 W_1 \, / \, d_2 W_0, \, 0)$ where W_1 and W_0 are the arrays

$$\begin{aligned} W_i &= \{\langle (3,3), B\rangle, \langle (3,4), B\rangle\} \\ W_0 &= \{\langle (0,0), R\rangle\} \end{aligned}$$

Let us suppose that d_2 belongs to w_0 and W_0 belongs to $\mathcal{E}$. In order to prove that r_1 is applicable, first we check that z_2 is contained in the cell 1 and, according to the previous claim, d_2 is contained in the environment.

We have several possibilities to choose the pair $\mathbf{z}_i, \mathbf{z}_0$. The different choices show the no determinism of the system. We also apply the rule with maximal parallelism. In this case we take the following option: the pair $\mathbf{z}_i, \mathbf{z}_0$ with $\mathbf{z}_i = (3, -2)$ and $\mathbf{z}_0 = (6, 1)$ for the first application of the rule and the pair $\mathbf{z}_i^*, \mathbf{z}_0^*$ with $\mathbf{z}_i^* = (5, -2)$ and $\mathbf{z}_0^* = (8, 2)$ for the second application.

(a) $W_1 + \mathbf{z}_i = \{\langle (6,1), B\rangle, \langle (6,2), B\rangle\} \subseteq A_1$
(a) $W_1 + \mathbf{z}_i^* = \{\langle (8,1), B\rangle, \langle (8,2), B\rangle\} \subseteq A_1$
(b) $supp(W_i + \mathbf{z}_i) \cap supp(W_0 + \mathbf{z}_0) = \{(6,1), (6,2)\} \cap \{(6,1)\} \neq \emptyset$
(b) $supp(W_i + \mathbf{z}_i^*) \cap supp(W_0 + \mathbf{z}_0^*) = \{(8,1), (8,2)\} \cap \{(8,1)\} \neq \emptyset$
(c) $supp(A_i \; - \; (W_i + \mathbf{z}_i)) \cap supp(W_0 + \mathbf{z}_0) = \{(5,2), (7,2), (8,2), (9,2), (8,1)\} \cap \{(6,1)\} = \emptyset$
(c) $supp(A_i \; - \; (W_i + \mathbf{z}_i)) \cap supp(W_0 + \mathbf{z}_0) = \{(5,2), (6,2)(7,2), (9,2), (861)\} \cap \{(8,2)\} = \emptyset$

The rule r_1 is applicable and the result of applying the rule twice is $[d_2^2 z_2 c_3 A'_1\,]_1$ where

$$\begin{aligned} A'_1 &= (A_1 \; - \; (W_1 + \mathbf{z}_1) - \; (W_1 + \mathbf{z}_1)^*) \cup (A_0 + \mathbf{z}_0) \cup (A_0 + \mathbf{z}_0^*) \\ &= \{\langle (5,2), R\rangle, \langle (7,2), G\rangle, \langle (9,2), R\rangle, \langle (6,1), R\rangle, \langle (8,2), R\rangle\} \end{aligned}$$

Finally, let us consider the case in which one of the regions involved in the rule is the environment and the array considered in the environment is the empty array.

Definition 4. *The communication rule* $(i, u_i W_i / u_0, 0)$ *is* applicable *over two cells labeled by* i *and* 0 *if the following conditions are verified:*

- u_i *is contained in cell* i *and* u_0 *is contained in cell* 0
- *There exist an array* A_i *in the cell* i *and a pair* $\mathbf{z}_i \in \mathbb{Z}^2$ *such that* $W_i + \mathbf{z}_i \subseteq A_i$

The application of this rule means that the objects of the multisets represented by u_i *is removed from the cell* i *and substituted by the multiset represented by* u_0*. The array* A_i *in the cell* i *is substituted by* A'_i *where* $A'_i = (A_i - (W_i + \mathbf{z}_i))$

Example 4. Let us suppose the cell 1 with the following objects and arrays, $[\, z_2^3 c_3 A_1 \,]_1$ and A_1 the array over $\{R, B, G\}$

$$\begin{aligned} A_1 = \{ \; & \langle (5,2), R\rangle, \langle (6,2), B\rangle, \langle (7,2), G\rangle, \langle (8,2), B\rangle\} \\ & \langle (9,2), R\rangle, \langle (6,1), B\rangle, \langle (8,1), B\rangle\} \end{aligned}$$

Let us consider the rule $r_1 \equiv (1, W_1 \,/\, d, 0)$ where W_1 is the array $W_i = \{\langle (3,3), B\rangle\}$. Let us suppose that d belongs to w_0. In this case, we have four possibilities to choose $\mathbf{z}_i$. They are $(3,-2), (3-1), (5,-2), (5,-1)$. It is trivial to check that the rule is applicable. It will be applied with maximal parallelism, so the rule will be applied four times and the result of applying the rule twice is $[d^4 z_2^3 c_3 A'_1\,]_1$ where

$$A'_1 = \{\langle (5,2), R\rangle, \langle (7,2), G\rangle, \langle (9,2), R\rangle\}$$

4 Using Array Tissue-like P Systems in Digital Image

In digital image terminology, given a finite *alphabet of colors* V and a *blank* object $\#$ such that $\# \notin V$, a two-dimensional (2D) digital image is a pair (S, A_S), where $S \subset \mathbb{N}^2$ and $A_S : S \to V \cup \{\#\}$ is an array on S. The size of V, $|V|$, is the number of its elements. Moreover, we can introduce an order of colors in an image. We define the *ordered alphabet associate to an image* like a pair $(V, <_V)$, where $<_V$ is an order in the set V.

The definition of *pixel* is associated with arrays, i.e., with equivalence classes of arrays. In this way, it makes sense that we study the adjoining relation of two pixels of generic positions (i, j) and (i', j') by exploring the relation among these generic coordinates. For the sake of simplicity, we write the pixel $< (i, j), a >$ as a_{ij}. There exists two natural way of defining adjacent pixels: 4-adjacency and 8-adjacency [17, 18].

In the first case, given a pixel K_{ij}, the list of adjacent pixels to this is $\{K_{ij-1}, K_{ij+1}, K_{i-1j}, K_{i+1j}\}$ i.e.; the adjacent pixels to any pixel K_{ij} are just north, south, west, east of this (no in the diagonal respect to considered pixel). In the second we consider the pixel K_{ij} (where $K = B \vee K = W$), the list of adjacent pixels to this is $\{K_{i-1j-1}, K_{i-1j}, K_{i-1j+1}, K_{ij-1}, K_{ij+1}, K_{i+1j-1}, K_{i+1j}, K_{i+1j+1}\}$ i.e.; the adjacent pixels to a any pixel K_{ij} are just up, down, right and left of this and, moreover, we consider the diagonal objects.

We will consider to work in this paper with 4-adjacency (for 2D images), because from a membrane computing point of view is more complex to design systems using this adjacency.

In this paper, we want to segment a 2D digital image. For this, we obtain the boundary of the different regions dividing the image. If we want to draw (or highlight) the boundaries we can follow two paths: *edge-based segmentation* and *region-based segmentation*. In the first option, we want to draw border line of the regions. In the second, we want to eliminate (or draw in white) and keep the resting pixels of the regions.

4.1 Segmenting 2D images

In this paper, we have decided to segment 2D digital images using array tissue-like P systems based in the first method: *edge-based segmentation*

Edge-based segmentation

We must find the border points of the regions (with different color) present in an image. So, we look for the pixels a_{ij} with some adjacent pixel of different color. We consider an input 2D digital image, and the color alphabet of the image ordered. So, for each image with $n \times m$ pixels ($n, m \in \mathbb{N}$) we define an array tissue-like P system whose input is given by an array codifying the input image. For the answer stage we use a counter $\bar{z}_i$, whose number of copies initially is $\lceil r^{1/2^7} \rceil$, where $r = max(n, m)$ because segmentation takes place in a constant number of steps.The output of the system is given by the objects appear in the output cell when it stops.

So, we can define an array tissue-like P systems to do the edge-based segmentation to a 2D image. For each $n, m \in \mathbb{N}$ we consider the tissue-like P system with input of degree 2

$$\Pi = (\Gamma, \Sigma, V, \mathcal{E}, w_0, w_1, w_2, A_1, A_2, \mathcal{R}, i_\Pi, o_\Pi),$$

defined as follows

(a) $\Gamma = \Sigma \cup \{a'_{ij} \,:\, a \in \mathcal{C}_S,\ 1 \leq i \leq n,\ 1 \leq j \leq m\} \cup \{\bar{z}_1\}$,
(b) $\Sigma = \{a_{ij} : a \in \mathcal{C}_S,\ 1 \leq i \leq n,\ 1 \leq j \leq m\}$,
(c) $V = \mathbb{N}$
(d) $\mathcal{E} = \{\, a'\,b\,,\, b\,a'\,, \begin{matrix} a' \\ b \end{matrix}, \begin{matrix} b \\ a' \end{matrix}, \begin{matrix} a' & b \\ a' & a' \end{matrix}, \begin{matrix} a' & a' \\ a' & b \end{matrix}, \begin{matrix} a' & a' \\ b & a' \end{matrix}, \begin{matrix} a' & a' \\ b & a' \end{matrix} \,:\, a, b \in V \,\wedge\, a < b\}$,
(e) $w_0 = \{\bar{z}_i \,:\, 2 \leq i \leq 9\}$,
$w_1 = \bar{z}_1^{\lceil r^{1/2^7} \rceil}$, where $r = max(n, m)$,
$w_2 = \bar{z}_1^{\lceil r^{1/2^7} \rceil}$,
(f) $A_1, A_2 = \emptyset$

(g) R is the following set of communication rules:

1. $$(j, \bar{z}_i/\bar{z}_{i+1}^2, 0),\text{ for } i = 1, \dots, 8,\ j = 1, 2$$

   In this rule, we are working with a counter that it is used in the output of the systems.

2. $$(1,\ a\ b\ /\ a'\ b\,, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   $$(1,\ b\ a\ /\ b\ a'\,, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   $$(1, \begin{matrix} a \\ b \end{matrix} / \begin{matrix} a' \\ b \end{matrix}, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   $$(1, \begin{matrix} b \\ a \end{matrix} / \begin{matrix} b \\ a' \end{matrix}, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   These rules are used when image has two adjacent pixels with different associated colors (border pixels). Then, the pixel with less associated color is marked (edge pixel).

3. $$(1, \begin{matrix} a' & b \\ a & a' \end{matrix} / \begin{matrix} a' & b \\ a' & a' \end{matrix}, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   $$(1, \begin{matrix} a' & a \\ b & a' \end{matrix} / \begin{matrix} a' & a' \\ b & a' \end{matrix}, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   $$(1, \begin{matrix} a & a' \\ a' & b \end{matrix} / \begin{matrix} a' & a' \\ a' & b \end{matrix}, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   $$(1, \begin{matrix} b & a' \\ a' & a \end{matrix} / \begin{matrix} b & a' \\ a' & a' \end{matrix}, 0),\text{ for } a, b \in \mathcal{C},\ a < b.$$

   The rules mark (write in capital letters) the pixels which are adjacent to two pixels with same color and which were marked before. But, with the condition that the marked objects are adjacent to other pixel with a different color.

4. $$(1, \bar{z}_9 a'_{ij}/\bar{z}_9, 2),\text{ for } a \in \mathcal{C}, 1 \le i \le n,\ 1 \le j \le m$$

   With these rules system sends the edge pixels to the output cell.

(h) $i_\Pi = 1$
(i) $o_\Pi = 2$.

An overview of the Computation: A 2D image is codified by the input array that appear in the input cell and with them the system begins to work. Rules of type 1 initiate the counter $\bar{z}$. In a parallel manner, rules of type 2 identify the border pixels and mark the edge pixels. These rules need 4 steps to mark all the border pixels. From the second step, the rules of type 3 can be used with the first rules at the same time. So, in other 4 steps we can mark the rest of the (edge) pixels adjacent to two edge pixels and other border pixel with a different color to the other three pixels. System can apply the types of rules 2 and 3 simultaneously in some configurations, but it always applies the same number of these two types of rules because this number is given by edge pixels (we consider 4-adjacency). Finally, the fourth type of rules are applied in the following step on the system finish to mark all edge pixels in the cell 1. So with one step more we will have all the edge pixels in the output cells. Thus we need only 9 steps to obtain an edge-based segmentation for an $n \times m$ digital image.

Examples

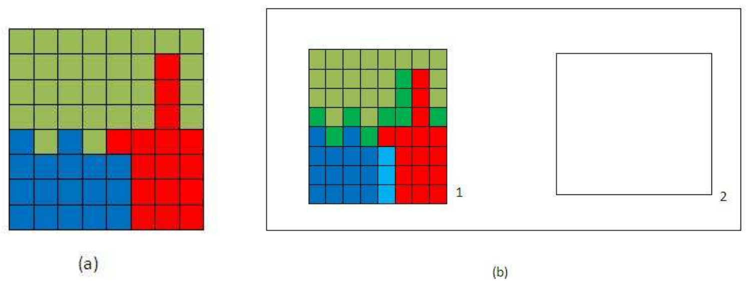


Fig. 2. (a) Input Image (b) Initial configuration

We show here some examples to see how our system does the edge-based segmentation of 2D images. We work with the images given by Figure 2 (a) and Figure 4 (a).

Initially, we consider an 8×8 image given by Figure 2 (a). A codifying of the initial configuration to segment this image is shown in Figure 2 (b). The order of the colors is the following: green, blue and red. Remember, we apply the rules in a maximally parallel manner where the pixels used by the rules are shown with different colors in the Figure 3 (a).

After nine steps, the output configuration is obtained and shown in Figure 3 (b).

Next, we segment the image of size 12×14 given by Figure 4 (a). In this example, we take the colors in the following order: Red, green, brown, orange, black, blue and light blue. The output image is shown in Figure 4(b).

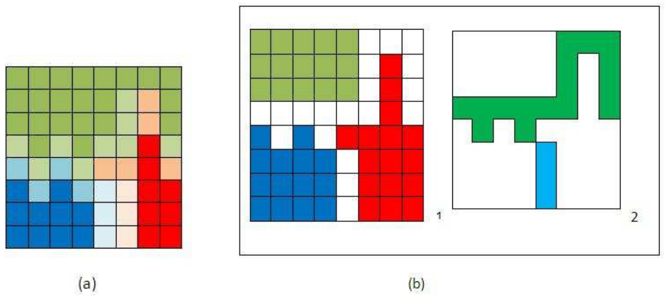


Fig. 3. (a) Process (b) Output Configuration

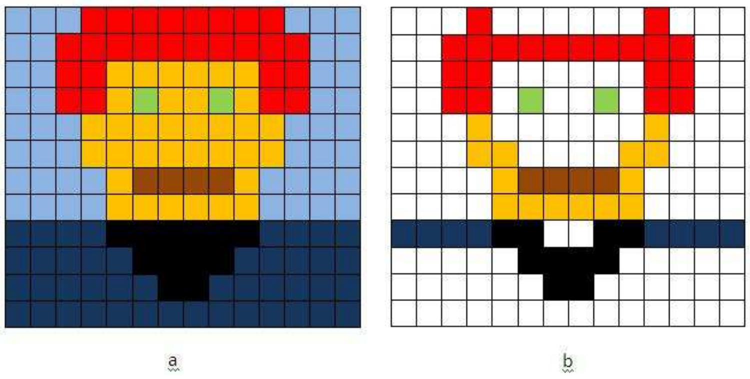


Fig. 4. (a) Input Image (b) Output Configuration

5 Final Remarks

This paper can be seen as a first attempt of formalizing the bridges between Membrane Computing and Algebraic Topology presented recently by Cristinal *et al.* [3, 4, 5].

The starting point is that problems from Digital Images, treated by techniques of Algebraic Topology, can be suitable for Membrane Computing techniques. The basis is that such problems can be treated locally by a set of processors, the information can be expressed as (multi)sets of pixels and other auxiliary objects, and the transformations can be processed by re-writing-type rules.

Many research lines are open. From the Membrane Computing point of view, we wonder whether tissue-like models is the most suitable or not, or in which way this formalization can be improved. From the Algebraic Topology point of view, the question is to find new representations and new problems which can be expressed and dealt with Membrane Computing techniques.

Acknowledgement

The first author acknowledge the support of the project ”Computational Topology and Applied Mathematics” PAICYT research project FQM-296. The second,third and four author acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. Ceterchi, R., Mutyam, M., Păun, Gh., Subramanian, K.G. Array-rewriting P Systems. *Natural Computing* **2**, (2003) 229-249.
2. Cook, C.R., Wang, P.S.-P. A Chomsky Hierarchy of Isotonic Array Grammars and Languages. *Computer Graphics and Image Processing* **8**, (1978) 144-152.
3. Christinal, H.A., Díaz-Pernil, D., Real, P. Segmentation in 2D and 3D Image Using Tissue-Like P System. *Lecture Notes in Computer Science*, **5856**, (2009) 169-176.
4. Christinal, H.A., Díaz-Pernil, D., Real, P. Using Membrane Computing for Obtaining Homology Groups of Binary 2D Digital Images. *Lecture Notes in Computer Science*, **5852**, (2009) 388-401.
5. Christinal, H.A., Díaz-Pernil, D., Real, P. P Systems and Computational Algebraic Topology. Accepted paper at *Journal of Mathematical and Computer Modelling.*
6. Freund, R. Array Grammars. Technical Rep. 15/00, Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, (2000).
7. Kim, S.-H., Kim, H.-G., Tchah, K.-H. Object oriented face detection using colour transformation and range segmentation, *Electronics Letters, IEEE*, **34**, (1998), 979-980.
8. Linda G. Shapiro and George C. Stockman .*Computer Vision*, (2001).
9. Martín–Vide, C., Pazos, J., Păun, Gh. and Rodríguez Patón, A. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science*, **2387**, (2002), 290–299.
10. Martín–Vide, C., Pazos, J., Păun, Gh. and Rodríguez Patón, A. Tissue P systems. *Theoretical Computer Science*, **296**(2), (2003), 295–326.
11. Păun, A. and Păun, Gh. The power of communication: P systems with symport/antiport. *New Generation Computing*, **20**, 3, (2002), 295-305.
12. Păun, Gh. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1, (2000), 108–143.
13. Păun, Gh. Membrane Computing. An Introduction. *Springer–Verlag, Berlin*, (2002).
14. Păun, Gh. Grammar Systems vs. Membrane Computing: A Preliminary Approach. Grammar Systems Week 2004, July 5-9, Budapest, Hungary, (2004). `http://psystems.disco.unimib.it/download/grsp0.ps`
15. Gh. Păun, G. Rozenberg, A. Salomaa, eds: Handbook of Membrane Computing. Oxford University Press, Cambridge, 2009.
16. Rosenfeld, A. Picture Languages. Academic Press, Reading, MA (1979).
17. Rosenfeld, A. Digital Topology. *American Mathematical Monthly*, **86**, (1979), 621-630.

18. Rosenfeld, A. Connectivity in Digital Pictures. *Journal for Association of Computing Machinery*, **17**(1), (1970), 146-160.
19. Sharmay, O., Miocz, D., Antony, F. Polygon Feature Extraction from Satellite imagery based on color image segmentation and medial axis. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **XXXVII**, (2008), 235-240.
20. Subramanian, K.G. P Systems and Picture Languages. *Lecture Notes in Computes Science* **4664**, (2007) 99-109.
21. Tobias, O.J., Seara,R. Image Segmentation by Histogram Thresholding Using Fuzzy Sets. *IEEE Transactions on Image Processing*,**11**, 12, (2002) 1457-1465.
22. Wang, P.S.-P. Some New Results on Isotonic Array Grammars. *Information Processing Letters* **10**, (1980) 129-131.
23. Wang, D., Lu, H., Zhang, J., Liang, J.Z. A Knowledge-Based Fuzzy Clustering Method with Adaptation Penalty for Bone Segmentation of CT images. *Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, (2005), 6488–6491.
24. Yuan Xiaojing, Situ Ning and Zouridakis George. A narrow band graph partitioning method for skin lesion segmentation.*Elsevier Science Pattern Recognition*, **42**, 6, (2009), 1017-1028.
25. Yazid. H and Arof. H. Image Segmentation using Watershed Transformation for Facial Expression Recognition. *IFMBE Proceedings, 4th Kuala Lumpur International Conference on Biomedical Engineering*, **21**, (2008), 575-578.
26. P systems web page `http://ppage.psystems.eu/`

Tissue-like P Systems Without Environment

Hepzibah A. Christinal[1], Daniel Díaz-Pernil[1],
Miguel A. Gutiérrez-Naranjo[2], Mario J. Pérez-Jiménez[2]

[1] Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics I
{hepzi,sbdani}@us.es

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
{magutier,marper}@us.es

University of Sevilla. Avda. Reina Mercedes s/n, 41012, Sevilla, Spain

Summary. In this paper we present a tissue-like P systems model with cell division the environment has been replaced by an extra cell. In such model, we present a uniform family of recognizer P systems which solves the Subset Sum problem. This solution establishes a new frontier for the tractability of computationally hard problems in Membrane Computing, since it proves that **NP**-complete problems can be solved without an arbitrarily large amount of objects in the environment.

1 Introduction

In Membrane Computing, the *environment* is the spatial location where the P system is placed. It appears in the description of all P system models in an explicit or implicit way. In this paper, we focus on its role in the framework of tissue-like P systems.

In cell-like models, it is defined as a region surrounding the skin (and therefore the whole P system) with no rules associated. Its role is inactive. It consists exclusively on holding objects, generally sent out by the P system. Occasionally, the objects in the environment can be sent into the P system if the skin has associated a send-in rule, but this is not the usual situation. If we consider the membrane structure of a cell-like P systems as a tree with the processor units (the membranes) on the nodes, the environment can be seen as a new node, linked uniquely with the skin and able to contain multisets of objects, but no rules.

The most common point of view is considering the cell-like P system as a black box where the computation takes place and where an external observer has no access. Such observer can only watch the skin and the surrounding region from a point out of the P system. Bearing in mind this point of view, the resulting product of the computation must be expelled to the environment in order to be observed.

In Spiking Neural P system, the environment is also considered the region surrounding the whole system. It does not belong to the formal description of the system, but it is implicitly considered, since one neuron is marked to send spikes *out* of the system.

In spite of the membrane structure of a SN P system is a general graph instead of a tree as in the cell-like model, they share a common property with respect to the environment. In both models only one membrane (neuron, in the usual terminology of SN P systems) is linked to the environment: In cell-like models, it is the skin and in the spiking model, it is the *output neuron*. Beyond this similarity, the role of the environment is even more restrictive in the case of the SN P systems. According to this model, the information is encoded in time, so the important question is to consider the moment in which the spikes are sent out by the output neuron. Such spikes are not stored and can be forgotten.

The role of the environment changes in tissue-like P systems [13, 14]. In such P systems, the cells are placed in a general graph[3], and, potentially, all of them can trade objects against the environment. The main feature of the environment is the arbitrarily large amount of objects placed in it. These objects can participate on the computation according to the symport/antiport rules associated to cells of the system. The biological inspiration it is clear, a living tissue can take from outside as much oxygen and nutrients as it needs without limitation.

This arbitrarily large amount of objects in the environment has been widely exploited in the design of efficient solutions to **NP**-problems by recognizer tissue-like P systems with cell division (see, e.g., [4, 5, 6]). In such designs, the initial resources of the devices are polynomial in the size of the input and the number of objects taken from the environment along the computation is not considered in the initial description.

From this starting point, it is natural to wonder if this singularity can be avoided. In other words, we wonder if tissue-like P systems in which environment is empty on the input can also solve **NP**-problems.

In this paper we give a positive answer to this question. We present a tissue-like P systems model with cell division where environment is supplied by a cell. To do this, we divide this cell so many time as we need. In this manner, we generate so copies of initial objects of this cell as we want.

In such model, we present a uniform family of recognizer tissue-like P systems which solves the Subset Sum problem. This solution establishes a new frontier for the tractability of computationally hard problems in Membrane Computing [8], since it proves that **NP**-complete problems can be solved without an arbitrarily large amount of objects in the environment.

Bearing in mind these considerations, if the initial amount of objects in the environment is fixed in a similar way to the cells, then the environment can be

[3] In fact, a *virtual* graph, as we will see below.

seen as a new cell w_0. The particular feature of this distinguished cell is that it cannot be divided.

The paper is organized as follows: In Section 2 we recall some basic concepts which will be used later. In Section 3 we present the model of tissue-like P systems without environment and cell division and in Section 4 a solution to the Subset Sum problem is this framework is shown. The paper finishes with some final remarks and comments on the future work.

2 Preliminaries

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

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

A *multiset* over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. The set of all multisets on A will be denoted by $\mathcal{M}(A)$. If $m = (A, f)$ is a multiset then its *s*upport is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite). If $m = (A, f)$ is a finite multiset over A, then it will be denoted as $m = \{\{a_1^{f(a_1)}a_2^{f(a_2)}\cdots a_k^{f(a_k)}\}\}$, where $supp(m) = \{a_1, \dots, a_k\}$, and for each element a_i, $f(a_i)$ is called the multiplicity of a_i. If $f(a_i) = 1$, we will write a_i instead of a_i^1. In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems[4].

3 Tissue-like P Systems without Environment

Tissue P systems were defined in [13, 14] under two biological inspirations: intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. From the initial definition, several research lines have been developed and other variants have arisen (see, for example, [2, 3, 7, 11, 12, 18]). Based on the cell-like model of P systems with active membranes, Gh. Păun et al. presented in [16] a new model of tissue P systems endowed with *cell division.* The biological inspiration is clear: alive tissues are not *static* network of cells, since cells are duplicated via mitosis in a natural way. In this model, the tissue (of cells) is formed by the cells and a region called *environment* containing all of them. Moreover, this model deals with

[4] We refer to [15] for basic information in this ares, to [17] for a comprehensive presentation and the web site [19] for the up-to-date information.

an arbitrarily large amount of objects in the environment, and it can not divided along a computation.

Next, we present a variant of this model, in which we drop one ingredient: the arbitrary large amount of objects in the environment. The key idea is to consider a set of initial cells $w_1, \ldots, w_n$ plus an extra cell w_0. This extra cell will have the same behavior as the other ones, but it will assume the role of the environment. As we pointed out above, the resources in this cell will be also computed as initial resources and must be polynomially generated.

Formally, a *tissue-like P system without environment* (or simplifying *tissue-like P system*$_{WE+D}$) of degree $q \geq 1$ is a tuple of the form

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

where:

1. Γ is a finite *alphabet*, whose symbols will be called *objects*.
2. $env(= w_0)$, is a string over Γ representing the multisets of objects associated with the environment in the initial configuration.
3. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells in the initial configuration.
4. $\mathcal{R}$ is a finite set of rules of the following form:
   (a) *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, \ldots, q\}, i \neq j,\ u, v \in \Gamma^*$ and 0 represents to the environment.
   (b) *Division rules*: $[a]_i \rightarrow [b]_i[c]_i$, where $i \in \{1, \ldots, q\}$ and $a, b, c \in \Gamma$. Note that the environment (labeled by 0) cannot divide.
5. $i_0 \in \{0, 1, 2, \ldots, q\}$ denotes the output region, which can be the environment ($i_0 = 0$) or the region inside a cell ($1 \leq i_0 \leq q$).

In tissue-like P systems, the graph structure of the cells is not given in an explicit way. The links between regions are provided by the set of symport/antiport rules. It is known as a *virtual graph.* In such way, two cells are linked if and only if there is a rule that allows the interchange of objects between them. In a similar way, any cell can trade objects against the environment if there exists a rule for this purpose. Notice that the rules are associated to the labels. In such way, the graph is dynamical, since new nodes can appear produced by the application of division rules.

The application of rules in this new model is the same as in usual tissue-like P systems with cell division:

- The communication rule $(i, u/v, j)$ can be applied over two regions i and j such that u is contained in cell i and v is contained in region j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells.
- The division rule $[a]_i \rightarrow [b]_i[c]_i$ is applied over a cell $i \in \{1, \ldots, q\}$ containing object a. The application of this rule divides this cell into two new cells with the same label. All the objects in the original cell are replicated and copied in

the new cell, with the exception of the object a, which is replaced by the object b in the first one and by c in the other one.

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

The cells obtained by division have the same labels as the original cell and if a cell is divided, its interaction with other cells is blocked during the mitosis process. In some sense, this means that while a cell is dividing it closes the communication channels with other cells.

A *configuration* is an instantaneous description of the system Π, and it is represented as a tuple $(w_0, w_1, \dots, w_q)$. Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A sequence of computation steps is called a *computation.* A configuration is *halting* when no rules can be applied to it. Then, a computation halts when the system reaches a halting configuration. In the literature, the output of a computation is collected from its halting configuration by reading the objects contained in the output cell.

3.1 Recognizer Tissue-like P Systems$_{WE+D}$

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

In order to study the computational efficiency for solving **NP**-complete decision problems, a special class of P systems were introduced in [1]: recognizer P systems. The original definition corresponds to *cell-like* P systems, but it was extended in a natural way in [16] to *tissue-like* ones.

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

In the literature, recognizer P systems are associated with P systems with *input* in a natural way. The data encoding to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane.* The output of the computation (`yes` or `no`) is sent to the output region, in the last step of the computation.

A recognizer tissue-like P system$_{WE+D}$ of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \Sigma, w_0, w_1, \dots, w_q, \mathcal{R}, i_{in}, i_0)$$

where

- $(\Gamma, w_0, w_1, \dots, w_q, \mathcal{R}, i_0)$ is a tissue-like P system$_{WE+D}$ of degree $q \geq 1$ (as defined in the previous section), $M(\sigma)$ is a string over $\Gamma \setminus \Sigma$, for each $\sigma \in V \cup \{w_0\}$.
- The working alphabet Γ has two distinguished objects `yes` and `no`, present in at least one copy in some initial multisets.
- Σ is an (input) alphabet strictly contained in Γ.
- $i_{in} \in \{1, \dots, q\}$ is the input cell.
- All computations halt.
- If $\mathcal{C}$ is a computation of Π, then either the object `yes` or the object `no` (but not both) must have been released into the output region, and only in the last step of the computation.

The computations of the system Π with input $w \in \Sigma^*$ start from a configuration of the form $(w_0, w_1, w_2, \dots, w_{i_{in}} \cup w_i, \dots, w_q)$, that is, after adding the multiset w to the contents of the input cell i_{in}.

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

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

We denote by $\mathbf{PMC}_{TD-E}$ the set of all decision problems which can be solved by means of recognizer tissue-like P systems$_{WE+D}$ in polynomial time. This class is closed under polynomial reduction and under complement.

4 A Solution for the Subset Sum Problem

The Subset Sum problem is very well-known. It can be settled as follows: *Given a finite set V, a weight function, $w : V \to \mathbb{N}$, and a constant $k \in \mathbb{N}$, determine whether or not there exists a subset $B \subseteq V$ such that $w(B) = k$.*

Next, we prove that the Subset Sum problem can be solved in a linear time (in $\{n, \log\, k\}$) by a family of recognizer tissue-like P systems$_{WE+D}$. An instance $u = (V, w, k)$ of the Subset Sum Problem with $V = \{v_1, v_2, \ldots, v_n\}$ will be represented by $u = (n, (w_1, \ldots, w_n), k)$, where $w_i = w(v_i)$, for each i $(1 \leq i \leq n)$. Such an instance will be encoded as the multiset $cod(u) = \{\{v_i^j : w(A_i) = j \wedge i \in \{1, \ldots, n\}\}\} \cup \{\{q^k\}\}$.

Next, we present a family of recognizer tissue-like P systems$_{WE+D}$ with cell division where at the initial configuration each system of the family has four regions (labeled by 0,1,2 and 3).

We will address the resolution via a brute force algorithm, which consists in the following stages:

- *generation stage*: all possible subsets of V are generated by successive application of division rules;
- *pre-checking stage*: the weight of each subset of V is calculated;
- *checking stage*: It is check if there exists a subset of V with weight equal to k;
- *output stage*: an affirmative or negative answer to the problem is given, according to the results of the previous stage.

For each $(n, k) \in \mathbb{N}^2$ we will consider the system

$$\Pi(n, k) = (\Gamma, \Sigma, V, env, L, M, \mathcal{R}, \mathcal{E}, i_{in}, i_0),$$

where

- $\Gamma = \Sigma \cup \{A_i, B_i : 1 \leq i \leq n\}$
  $\cup\ \{G_i : 1 \leq i \leq n + \lceil \log(k+1) \rceil - 2\}$
  $\cup\ \{a_i : 1 \leq i \leq 2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 9\}$
  $\cup\ \{\bar{c}_i : 1 \leq\ i \leq n + \lceil \log(k+1) \rceil - 1\}$
  $\cup\ \{c_i : 1 \leq\ i \leq n + 1\}$
  $\cup\ \{d_i : 1 \leq i \leq \lceil \log n \rceil + \lceil \log(k+1) \rceil + 3\}$
  $\cup\ \{e_i : 1 \leq i \leq \lceil \log n \rceil + 1\}$
  $\cup\ \{B_{ij} : 1 \leq i \leq n\ \wedge\ 1 \leq j \leq \lceil log(k+1) \rceil + 1\}$
  $\cup\ \{\alpha, b, D, p, q, g_1, g_2, f_1, T, S, N, \texttt{yes}, \texttt{no}\}$
- $V = \{\sigma_1, \sigma_2, \sigma_3\} \cup \{env\}$
- $\Sigma = \{v_i : 1 \leq i \leq n\} \cup \{q\}$
- $L(\sigma_1) = 1, L(\sigma_2) = 2, L(\sigma_3) = 3, L(env) = 0$
- $M(\sigma_1) = a_1\, b\, \bar{c}_1\, \texttt{yes}\, \texttt{no}$
- $M(\sigma_2) = D\, A_1 \ldots A_n$

- $M(\sigma_3) = \{\{G_1 \dots G_{n+\lceil\log(k+1)\rceil-2}\, c_2^2 \dots c_{n+1}^2\, e_1\, e_2^2 \dots e_{\lceil\log n\rceil+1}^2$
  $d_1 \dots d_{\lceil\log n\rceil+\lceil\log(k+1)\rceil+3}\, p^k\, T\, S\, N\, g_1\, g_2\, f_1\}\}$
  $\cup\ \{\{B_{i1}\ 1 \le i \le n\}\}$
  $\cup\ \{\{B_{ij}^2\ 1 \le i \le n \wedge 2 \le j \le \lceil log(k+1)\rceil + 1\}\}$
- $M(env) = a_2\ \dots a_{2n+\lceil\log n\rceil+2\lceil\log(k+1)\rceil+9}\, \bar{c}_2\ \dots \bar{c}_{n+\lceil\log(k+1)\rceil-1}\, c_1$
- $\mathcal{R}$ is the following set of rules:
  1. *Division rules:*
     $r_{1,i} \equiv [G_i]_3\ \rightarrow\ [\alpha]_3[\alpha]_3$ for $i = 1, \dots, n + \lceil\log(k+1)\rceil - 2$
     $r_{2,i} \equiv [A_i]_2\ \rightarrow\ [B_i]_2[\alpha]_2$ for $i = 1, \dots, n$
  2. *Communication rules:*
     $r_{3,i} \equiv (1, a_i/a_{i+1}, 0)$ for $i = 1, \dots, 2n + \lceil\log n\rceil + 2\lceil\log(k+1)\rceil + 8$
     $r_{4,i} \equiv (1, \bar{c}_i/\bar{c}_{i+1}, 0)$ for $i = 1, \dots, n + \lceil\log(k+1)\rceil - 2$
     $r_5 \equiv (1, \bar{c}_{n+\lceil\log(k+1)\rceil-1}/c_1, 0)$
     $r_{6,i} \equiv (1, c_i/c_{i+1}^2, 3)$ for $i = 1, \dots, n$
     $r_7 \equiv (1, c_{n+1}/D, 2)$
     $r_8 \equiv (2, c_{n+1}/d_1 e_1, 3)$
     $r_{9,i} \equiv (2, e_i/e_{i+1}^2, 3)$ for $i = 1, \dots, \lceil\log n\rceil$
     $r_{10,i} \equiv (2, d_i/d_{i+1}, 3)$ for $i = 1, \dots, \lceil\log n\rceil + \lceil\log(k+1)\rceil + 2$
     $r_{11,i} \equiv (2, e_{\lceil\log n\rceil+1}B_i/B_{i1}, 3)$ for $i = 1, \dots, n$
     $r_{12,i,j} \equiv (2, B_{ij}/B_{ij+1}^2, 3)$ for $i = 1, \dots, n,\ j = 1, \dots, \lceil\log(k+1)\rceil$
     $r_{13,i} \equiv (2, B_{i\lceil\log(k+1)\rceil+1}v_i/p, 3)$ for $i = 1, \dots, n$
     $r_{14} \equiv (2, pq/\lambda, 0)$
     $r_{15} \equiv (2, d_{\lceil\log n\rceil+\lceil\log(k+1)\rceil+3}/g_1 f_1, 3)$
     $r_{16} \equiv (2, f_1 p/\lambda, 0)$
     $r_{17} \equiv (2, f_1 q/\lambda, 0)$
     $r_{18} \equiv (2, g_1/g_2, 3)$
     $r_{19} \equiv (2, g_2 f_1/T, 3)$
     $r_{20} \equiv (2, T/\lambda, 1)$
     $r_{21} \equiv (1, bT/S, 3)$
     $r_{22} \equiv (1, S\texttt{yes}/\lambda, 0)$
     $r_{23} \equiv (1, a_{2n+\lceil\log n\rceil+2\lceil\log(k+1)\rceil+9}b/N, 3)$
     $r_{24} \equiv (1, N\texttt{no}/\lambda, 0)$

- $i_{in} = 2$, is the input cell
- $i_0 = 0$, is the output cell

4.1 An Overview of the Computation

First of all, we recall the polynomial encoding of the Subset Sum problem in the family Π constructed in the previous section. Let $u = (n, (w_1, \dots, w_n), k)$ be an instance of the problem, $s(u) =< n, k >$ and $cod(u) = \{\{v_i^j\ :\ w(A_i) = j\ \wedge\ 1 \le i \le n\}\} \cup \{\{q^k\}\}$.

Next, we describe informally how the recognizer tissue P system with cell division $\Pi(s(u))$ with input $cod(u)$ works. Let us start with the *generation stage.*

Recall that if a division rule is triggered, the communication rules cannot be simultaneously applied. In this stage we have three parallel processes:

- The first one occurs in the region labeled by 1, where we have two counters: a_i, which will be used in the answer stage, and $\bar{c}_i$, which will be used to delay the start of the communication rules.
- The second one occurs in the region labeled by 2, where the second group of division rules are applied. For each object A_i (which codifies a member of the set V) we obtain two cells labeled by 2: One of them has an element B_i and the other one has an object α. Such object will not be used any more in the computation.
- The third one occurs in the cell labeled by 3, where the first group of division rules are applied for $n + \lceil \log(k+1) \rceil - 2$ steps. For each object G_i, we obtain two cells labeled by 3: both of them have an object α.

When all divisions have been done, we will have 2^n cells with label 2, in which each one of them will contain the encoding of a subset of V and $2^{n+\lceil \log(k+1) \rceil -2}$ cells with label 3.Then $\bar{c}_{n+\lceil \log(k+1) \rceil}$ is replaced by c_1 in cell 1. After this step, c_i will be multiplied until getting exactly 2^n copies in n steps. At this moment, the generation stage ends and the pre-checking stage begins.

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

(a) The counter d_i lets the checking stage start, since it produces the occurrence of the objects g_1 and f_1 in cells 2.
(b) The counter e_i will be multiplied for obtaining n copies of $e_{\lceil \log n \rceil +1}$ in the step $n + \lceil \log n \rceil + 5$ from cell 3. Then, we trade objects $e_{\lceil \log n \rceil +1}$ and B_i against B_{i1} for each element A_i in the subset codifying a possible solution associated with the membrane. After that, for each $1 \leq i \leq n$ we get $k+1$ copies of $B_{i\lceil \log(k+1) \rceil +1}$ from cell 3. Then for each element A_i, we get w_i copies of object p, in the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 3$.

The checking takes place in the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 4$, when all pairs of objects p and q from any cell labeled by 2 are sent to the environment. In this way, if the weight of the subset associated with a cell is equal to k, then no object p or q remains in this cell in the next step. Otherwise, if the encoding is not exactly of weight k, then at least one object p or q will remain in the cell. In the next step the answer stage starts. Two cases must be considered for each cell:

- If no object p or q remains in the cell, the object f_1 keeps in the cell, g_1 evolves to g_2, and in the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 6$ the objects f_1 and g_2 are traded against T from the cell three. In the next step T is sent to the cell 1, and in the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 8$, the objects T and b are sent to the cell labeled by 3 traded against S. Finally in the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 9$ the objects S and `yes` are sent to the environment.

- If any object p or q remains in the cell, such object is sent to the environment together with the object f_1. This causes that the object b still remains in the cell 1 after the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 8$. In this way, the objects b and $a_{2n+\lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 9}$ are traded against the object N with the cell labeled 3, and in the step $2n + \lceil \log n \rceil + 2\lceil \log(k+1) \rceil + 10$ the objects N and `no` are sent to the environment.

4.2 Some Technical Considerations

In order to establish that the family $\mathbf{\Pi}$ is polynomially uniform by deterministic Turing machines we firstly note that the sets of rules associated with the systems $\Pi(n,k)$ are recursively defined. Hence, it suffices to justify that the amount of necessary resources for defining the systems is polynomial in $\max\{n, \lceil \log k \rceil\}$.

- Size of the alphabet: $n \cdot \lceil \log(k+1) \rceil + 8n + 5\lceil \log(k+1) \rceil + 3\lceil \log n \rceil + 30 \in O(n \cdot \log k)$
- Initial number of cells: $4 \in \theta(1)$.
- Initial number of objects: $5n + 3\lceil \log n \rceil + 3\lceil \log(k+1) \rceil + 31 \in \theta(n)$.
- Number of rules: $n \cdot \lceil \log(k+1) \rceil + 6n + 3\lceil \log(k+1) \rceil + 3\lceil \log n \rceil + 27 \in O(n \cdot \log k)$
- Maximal length of a rule: 3.

So, we can claim the following result.

Theorem 1. *SS* $\in \mathbf{PMC}_{TD-E}$

Taking into account that SS is an **NP**–complete problem, and that the class $\mathbf{PMC}_{TD-E}$ is closed under complement, the following is deduced.

Corollary 1. $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{TD-E}$

5 Conclusions and Future Work

The search of biologically inspired frontiers for tractability has been an active research area in the last years. Since the problem **P** vs. **NP** is still open and it seems that will remain open for a long time, the research faces the problem of finding new frontiers between these classes of problems. Current research on complexity in Membrane Computing focuses on looking for the minimum amount of ingredients of a P system model able to solve a **NP**-complete problem.

One of these steps was the discovery of the role of the dissolution rules (a rule apparently innocent) as the key stone for solving **NP**-complete problems in the framework of P systems with active membranes [10, 9].

In this paper we give a new step in the same direction. We have prove that the use of an arbitrarily large amount of objects in the environment can be removed from tissue-like P systems with cell division in order to solve **NP**-complete problems. The next steps in this research area will try to reduce the initial ingredients in order to make the frontier of tractability thinner and thinner.

Acknowledgement

The first author acknowledge the support of the project "Computational Topology and Applied Mathematics" PAICYT research project FQM-296. The second,third and four author acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. M.J. Pérez–Jiménez, A. Romero-Jiménez, F. 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 DCFS 2003*, (2003), 284–294.
2. Alhazov, A., Freund, R. and Oswald, M. Tissue P Systems with Antiport Rules ans Small Numbers of Symbols and Cells. *Lecture Notes in Computer Science* **3572**, (2005), 100–111.
3. Bernardini, F. and Gheorghe, M. Cell Communication in Tissue P Systems and Cell Division in Population P Systems. *Soft Computing*, **9**(9), (2005), 640–649.
4. Díaz-Pernil, D. *Sistemas P de Tejido: Formalización y Eficiencia Computacional.* PhD Thesis, University of Seville, (2008).
5. Díaz-Pernil, D., Gutiérrez, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A. Solving Subset Sum in linear time by using tissue P systems with cell division. *Lecture Notes in Computer Science*, **4527**, (2007), 170-179.
6. Díaz-Pernil, D., Gutiérrez, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A. A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, **404**, (1-2), (2008), 76-87.
7. Freund, R., Păun, Gh. and Pérez-Jiménez, M.J. Tissue P Systems with channel states. *Theoretical Computer Science*, **330**, (2005), 101–116.
8. Gutiérrez-Escudero, R., Pérez-Jiménez, M.J., M. Rius-Font. Characterizing tractability by tissue-like P systems. Lecture Notes in Computer Science, **5957**, (2010), 289-300.
9. Gutiérrez, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J. *Lecture Notes in Computer Science*, **3850**, (2006), 224-240.
10. Gutiérrez, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics*, **83**, (7), (2006) 593 - 611.
11. Krishna, S.N., Lakshmanan K. and Rama, R. Tissue P Systems with Contextual and Rewriting Rules. *Lecture Notes in Computer Science* **2597**, (2003), 339–351.
12. Lakshmanan K. and Rama, R. On the Power of Tissue P Systems with Insertion and Deletion Rules. In A. Alhazov, C. Martín-Vide and Gh. Păun (eds.) *Pre-proceedings of the Workshop on Membrane Computing*, (2003), 304-318.
13. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science* **2387**, (2002), 290–299.
14. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. Tissue P systems. *Theoretical Computer Science*, **296**, (2003), 295–326.

15. Păun, G. *Membrane Computing. An Introduction.* Springer–Verlag, Berlin, (2002).
16. Păun, G., Pérez-Jiménez, M.J. and Riscos-Núñez, A. Tissue P System with cell division. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez and F. Sancho-Caparrini (eds.), *Second Brainstorming Week on Membrane Computing*, (2004), 380–386.
17. Păun, G. Rozenberg, A. Salomaa, eds: Handbook of Membrane Computing. Oxford University Press, (2009).
18. Prakash, V.J. On the Power of Tissue P Systems Working in the Maximal-One Mode. In A. Alhazov, C. Martín-Vide and Gh. Păun (eds.). *Pre-proceedings of the Workshop on Membrane Computing*, (2003), 356-364.
19. P systems web page `http://ppage.psystems.eu/`

PCol Automata: Recognizing Strings with P Colonies*

Luděk Cienciala[1], Lucie Ciencialová[1], Erzsébet Csuhaj-Varjú[2,3], György Vaszil[2]

[1] Institute of Computer Science, Silesian University in Opava
Bezručovo nám. 13, 74601 Opava, Czech Republic
{ludek.cienciala,lucie.ciencialova}@fpf.slu.cz

[2] Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu

[3] Department of Algorithms and Their Applications
Faculty of Informatics, Eötvös Loránd University
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary

Summary. We introduce the concept of a P colony automaton, an automata-like construct combining properties of finite automata and P colonies. We present some preliminary results on the accepting power of several variants of these extremely simple language recognizing devices, and propose problems for future research.

1 Introduction

P colonies are particular variants of very simple tissue-like membrane systems, modeling a community of very simple cells living together in a shared environment (for P colonies, see [12, 13], for membrane computing we refer to [15, 16]. In the basic model, the cells, the basic computing agents, are represented by a 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 function of the system. Number k is said to be the capacity of the P colony. 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 the fact that an internal object c is sent out of the cell, to the environment, in exchange of the object d, which was present in the environment. Thus, after applying these rules in parallel, a cell containing the objects a, c will contain the objects b, d. With each cell, a set of programs composed

* Research supported in part by the Czech Science Foundation, "GAČR", project 201/09/P075 and by the Hungarian Scientific Research Fund, "OTKA", project K75952.

of rules is associated. In the case of P colonies of capacity k, each program has k rules; the rules of the program must be applied in parallel to the objects in the cell.

The cells of a P colony execute a computation by synchronously applying their programs to objects inside the cells and outside in the environment. 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 present in the environment.

P colonies have been extensively examined during the years: among other things, it has been shown that these extremely simple constructs are computationally complete computing devices even with very restricted size parameters and with other syntactic or functioning restrictions [1, 2, 3, 4, 5, 6, 9, 10].

In the generic model, the environment is a multiset of objects, and thus its impact 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 is augmented with a string put on an input tape to be processed by the P colony. These string corresponds to the impulse sequence coming from the environment. In addition to their rewriting rules and the rules for communicating with the environment, the cells have so-called tape rules which are used for reading the next symbol on the input tape. This is done by changing one of objects inside the cell 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. It is easy to observe that the model, called a P colony automaton or a PCol automaton, resembles to standard finite automata and P automata [7], furthermore, to colonies of formal grammars [11].

PCol automata may work in several computation modes: for example, at any step of the computation a maximal set of components may be active and each component (at least one component, or a maximal number of components) should perform a tape rule. These computation modes are the so-called t, *tmin*, and *tmax* modes. In some other cases, transitions, i.e., simultaneous applications of non-tape rules are also allowed. These cases are the so-called *nt*, *ntmin*, *ntmax*, and *initial* computation modes. 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).

Due to their extreme simplicity, it is a challenging question how much accepting power can be obtained by the different variants of PCol automata. In this paper we present some preliminary results. Among other things, we show that PCol automata working in any of the *nt*, *ntmin*, or *ntmax* computational modes are able to recognize every recursively enumerable language over any alphabet (thus over any unary alphabet as well). Notice that P colonies are able to generate/accept any recursively enumerable set of numbers, which set can be represented as the length set of words of a recursively enumerable language over a unary alphabet. The large recognizing power of PCol automata working in these modes is due to the unbounded "workspace" provided by the symbols sent to the environment by the components while performing n, nt, $ntmin$, or $ntmax$-transitions, respectively. In the case of t-mode, we have some preliminary results. It is shown that PCol automata are able to accept any regular language (over any alphabet). Furthermore, there is a PCol automaton which recognizes the non-context-free context-sensitive language $\{a^n b^n c^n \mid n \geq 1\}$. In the case of initial mode, we provide a PCol automaton which accepts the language $L = \{a^{2^n}\}$. Finally, we propose some research areas for future study.

2 Preliminaries and definitions

Let V be a finite alphabet, let the set of all words over V be denoted by V^*, and let ε be the empty word. We denote the number of occurrences of a symbol $a \in V$ in w by $|w|_a$.

If the set of non-negative integers is denoted by $\mathbb{N}$, then a multiset over a set V is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $\mathit{supp}(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, $\mathit{supp}(M) = \emptyset$. We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ε will represent the empty multiset which is also denoted by $\emptyset$.

We say that $a \in M$ if $M(a) \geq 1$, and the cardinality of M, $card(M)$ is defined as $card(M) = \Sigma_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \to \mathbb{N}$, $M_1 \subseteq M_2$ holds, if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and M_2 is defined as $(M_1 \cup M_2) : V \to \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all $a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$.

Now we define the notion of a PCol automaton.

Definition 1. A *PCol automaton* of capacity k and with n cells, $k, n \geq 1$, is a construct $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ where V is an *alphabet*, the alphabet of the PCol automaton, its elements are called *objects*; $e \in V$ is the *environmental object* of the automaton; $w_E \in (V - \{e\})^*$ is a string representing the multiset of objects different from e which is found in the environment initially; $(w_i, P_i), 1 \leq i \leq n$, is the i-th *cell*; and F is a set of *accepting configurations* of the PCol automaton.

For each cell, (w_i, P_i), $1 \leq i \leq n$, w_i is a multiset over V, it determines the initial contents of the cell, and its cardinality $|w_i| = k$ is called the *capacity* of the system; P_i is a set of *programs*, where every program is formed from k rules of the following types:

- *tape rules* of the form $a \xrightarrow{T} b$, or $a \overset{T}{\leftrightarrow} b$, called rewriting tape rules and communication tape rules, respectively; or

- *nontape rules* of the form $a \to b$, or $c \leftrightarrow d$, called rewriting (nontape) rules and communication (nontape) rules, respectively.

For each $i,\ 1 \leq i \leq n$, the set of *tape programs* is denoted by P_i^T, they are formed from one tape rule and $k-1$ nontape rules, the set of *nontape programs* which contain only nontape rules, is denoted by P_i^N, thus, $P_i = P_i^T \cup P_i^N,\ P_i^T \cap P_i^N = \emptyset$.

The *computation* of a PCol automaton starts in the initial configuration, and the configurations are changed by the cells with the application of some of their programs. The programs either change the objects inside the cells (with rewriting rules) or exchange them for other objects with the environment (with communication rules). During the computation, the PCol automaton processes an input word. The leftmost symbol of the yet non-read part of the input word is read during a configuration change if at least one cell applies a tape program which introduces the same symbol inside the cell as the symbol to be read either by rewriting or by communication.

A *configuration* of PCol automaton is an $(n+2)$-tuple $(u; u_E, u_1, \ldots, u_n)$, where $u \in V^*$ is the unprocessed (unread) part of the input string, $u_E \in (V - \{e\})^*$ represents the multiset of objects different from e in the environment, and $u_i, \in V^*,\ 1 \leq i \leq n$, represents the contents of i-th cell. The *initial configuration* is given by $(w; w_E, w_1, \ldots, w_n)$, the input word to be processed by the system and the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as configurations of the form $(\varepsilon; v_E, v_1, \ldots, v_n)$.

To describe the computation process formally, we introduce the following notation. For any rule r we define four mappings as follows. Let $X \in \{T, \varepsilon\}$, and if $r = a \overset{X}{\to} b$, then let $left(r) = a$, $right(r) = b$, $export(r) = \varepsilon$, and $import(r) = \varepsilon$; if $r = a \overset{X}{\leftrightarrow} b$, then let $left(r) = \varepsilon$, $right(r) = \varepsilon$, $export(r) = a$, and $import(r) = b$ for $b \neq e$, or $import(r) = \varepsilon$ for $b = e$. Let us extend this notation also for programs. For $\alpha \in \{left, right, export, import\}$ and for any program p, let $\alpha(p) = \bigcup_{r \in p} \alpha(r)$ where for a rule r and program $p = \langle r_1, \ldots, r_k \rangle$, the notation $r \in p$ denotes the fact that $r = r_j$ for some $j,\ 1 \leq j \leq q$. Moreover, for any tape program p containing the tape rule $r \in p$, we also define the mapping $read(p)$ as $read(p) = right(r)$ if r is a rewriting tape rule, or $read(p) = import(r)$ if r is a communication tape rule.

Let the programs of each P_i be labeled in a one-to-one manner by labels from the set $lab(P_i)$, $1 \leq i \leq n$, $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$. In the following, for the sake of brevity, if no confusion arises, we designate programs and their labels with the same letters, thus, for a label $p \in lab(P_i)$, we also write $p \in P_i$.

Let $c = (u; u_E, u_1, \ldots, u_n)$ be a configuration of a PCol automaton Π. We call a *set of programs*, P_c, *applicable in configuration* c, if the following conditions hold.

- If $p, p' \in P_c, p \neq p'$ and $p \in P_i, p' \in P_j$, then $i \neq j$;
- for each $p \in P_c$, if $p \in P_i$ then $left(p) \cup export(p) = u_i$;
- for each $p \in P_c$, if p is a tape rule, then $read(p) = a$ where $u = au'$;
- $\bigcup_{p \in P_c} import(p) \subseteq u_E$.

A configuration $c = (au; u_E, u_1, \dots, u_n)$, $a \in V$, is *changed* to a configuration $c' = (u'; u'_E, u'_1, \dots, u'_n)$ by applying the set P_c of applicable programs if the following properties hold:

- If there is a $p \in P_c$ such that $p \in P_i$, then $u'_i = right(p) \cup import(p)$, otherwise $u'_i = u_i$, $1 \leq i \leq n$;
- $u_E = u_E - \bigcup_{p \in P_c} import(p) \cup \bigcup_{p \in P_c} export(p)$; and
- if there is a tape program $p \in P_c$ with $read(p) = a$, then $u' = u$, otherwise $u' = au$.

We say that a set P_c of applicable programs is *maximal* (with respect to a certain additional property), if for any $p' \in \bigcup_{i=1}^{n} P_i$ (having the same additional property) such that $p' \notin P_c$, the set of programs $P_c \cup \{p'\}$ is not applicable.

Based on the properties of P_c, the set of programs applied to a configuration $c = (au; u_E, u_1, \dots, u_n)$, $a \in V$, we distinguish the following types of transitions. Let the configuration obtained after the application of P_c be denoted by $c' = (u'; u'_E, u'_1, \dots, u'_n)$. We have a

- *t-transition*, denoted by $\Rightarrow_t$, if $u' = u$ and P_c is maximal set of programs with respect to the property that every $p \in P_c$ is a tape program with $read(p) = a$;
- *tmin-transition*, denoted by $\Rightarrow_{tmin}$, if $u' = u$ and P_c is maximal set of programs with at least one $p \in P_c$, such that p is a tape program with $read(p) = a$;
- *tmax-transition*, denoted as $\Rightarrow_{tmax}$, if $u' = u$ and $P_c = P_T \cup P_N$ where P_T is a maximal set of applicable tape programs with $read(p) = a$ for all $p \in P_T$, the set P_N is a set of nontape programs, and $P_c = P_T \cup P_N$ is maximal;
- *n-transition*, denoted by $\Rightarrow_n$, if $u' = au$ and P_c is maximal set of nontape programs.

A PCol automaton works in the t (*tmax*, *tmin*) *mode of computation* if it uses only *t*- (tmax-, tmin-) transitions. It works in the *nt* (*ntmax* or *ntmin*) mode if at any computation step it may use a t- (tmax- or tmin-) transition or an n-transition,

A special case of the *nt* mode is called *initial*, denoted by *init*, if the computation of the automaton is divided in two phases: first it reads the input strings using t-transitions and after reading all the input symbols it uses n-transitions to finish the computation.

Let us designate $M = \{t, nt, tmax, ntmax, tmin, ntmin, init\}$. The *language accepted by a PCol automaton Π* as above is defined as the set of strings which can be read during a successful computation:

$$\begin{aligned} L(\Pi, mode) = \{w \in V^* | (w; w_E, w_1, \dots, w_n) & \text{ can be} \\ \text{transformed by } \Pi \text{ into } (\varepsilon; v_E, v_1, \dots, v_n) \in F & \\ \text{with a computation in mode } mode \in M\}. & \end{aligned}$$

Let $\mathcal{L}(PColA, mode)$ denote the class of languages accepted by PCol automata in the computational mode $mode \in M$, and let RE denote the class of recursively enumerable languages.

Now we demonstrate the above defined notions by an example.

Example 1. Let $\Pi = (\{a, b, c\}, e, w_E, (w_1, P_1), (w_2, P_2), (w_3, P_3), F)$ be a PCol automaton, where the sets of programs are defined as

P_1	P_2	P_3
$p_1 : \langle e \xrightarrow{T} b; e \leftrightarrow a \rangle$,	$p_A : \langle e \overset{T}{\leftrightarrow} a; e \leftrightarrow a \rangle$,	$p_I : \langle e \xrightarrow{T} a; e \leftrightarrow a \rangle$,
$p_2 : \langle e \xrightarrow{T} a; e \rightarrow a \rangle$,	$p_B : \langle e \overset{T}{\leftrightarrow} b; e \leftrightarrow a \rangle$,	$p_{II} : \langle e \xrightarrow{T} a; e \rightarrow b \rangle$,
$p_3 : \langle e \leftrightarrow a; e \leftrightarrow e \rangle$,	$p_C : \langle e \rightarrow a; e \leftrightarrow e \rangle$,	$p_{III} : \langle e \rightarrow a; e \leftrightarrow e \rangle$,

and $c = (w; aa, ee, ee, ee)$ is the current configuration of Π.

If $w = bw'$ for some $b \in V,\ w' \in V^*$, then Π can execute a t-transition by applying the set of programs $P_c = \{p_1, p_B\}$ since the third cell has no applicable tape program. For a tmax-transition, Π can apply the set $P_c = \{p_1, p_B, p_{III}\}$, in this case the third cell can use its applicable nontape program. For a tmin-transition, Π has to choose one from three possible sets of programs $\{p_1, p_B, p_{III}\}$, $\{p_1, p_C, p_{III}\}$, or $\{p_3, p_B, p_{III}\}$. For an n-transition, Π has to use the set $P_c = \{p_3, p_C, p_{III}\}$.

If $w = aw',\ a \in V,\ w' \in V^*$, then the sets of applicable programs for the different transition types are given in the following table.

transition types	applicable sets of programs
$t,\ tmax$	$\{p_2, p_A, p_{II}\}$
$tmin$	$\{p_2, p_A, p_{III}\}, \{p_2, p_C, p_I\}, \{p_2, p_C, p_{II}.\}, \{p_2, p_C, p_{III}\}, \{p_3, p_A, p_{II}\}, \{p_3, p_A, p_{III}\}, \{p_3, p_C, p_I\}, \{p_3, p_C, p_{II}\}$
n	$\{p_3, p_C, p_{III}\}$

If $w = cw',\ c \in V,\ w' \in V^*$, then there is no cell with an applicable tape program. The only set of applicable programs is the set $P_c = \{p_3, p_C, p_{III}\}$ for an n-transition.

Example 2. Let $L \subseteq \Sigma^*$ be a regular language, and let $M = (\Sigma, Q, \delta, q_0, F)$ be a finite automaton with $L(M) = L$, with alphabet Σ, set of states Q, initial state q_0, set of final states F, and transition function $\delta : \Sigma \times Q \rightarrow Q$.

It is not difficult to see that the PCol automaton $\Pi = (\Sigma \cup Q, e, (w, P), F')$ of capacity two simulates the computation of M, with initial cell contents $w = aq_0$ for some $a \in \Sigma$, set of rules

$$P = \{\langle x \xrightarrow{T} a,\ q \rightarrow q' \rangle \mid \text{for all } x \in \Sigma \text{ such that } \delta(x, q) = q' \text{ for some } q, q' \in Q\},$$

and set of final configurations

$$F' = \{(\varepsilon; \varepsilon, xq_f) | \text{for all } x \in \Sigma, \text{ and } q_f \in F\}.$$

Since P contains only tape programs, Π cannot execute any n-transitions, and since it has only one cell, $L(\Pi, t) = L(\Pi, tmax) = L(\Pi, tmin) = L(M)$.

3 PCol automata computing in the modes with n-transitions

In this section we show that if we consider the functioning modes which allow n-transitions at arbitrary points of the computational process, then PCol automata characterize the class of recursively enumerable languages. First we recall the notion of a two-counter machine which will be used in the proof.

A *two-counter machine*, see [8], $M = (\Sigma \cup \{Z, B\}, E, R, q_0, q_F)$ is a 3-tape Turing machine where Σ is an *alphabet*, E is a set of *internal states* with $q_0, q_F \in E$ being the initial and the final states, and R is a set of *transition rules.* The machine has a read-only input tape and two semi-infinite storage tapes which are used as counters. The alphabet of the storage tapes contains only two symbols, Z and B (blank), while the alphabet of the input tape is $\Sigma \cup \{B\}$. The symbol Z is written on the first, leftmost cells of the storage tapes which are scanned initially by the tape heads. An integer t can be stored by moving a tape head t cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. The machine is capable of checking whether a stored value is *zero* or not by looking at the symbol scanned by the tape heads. If the scanned symbol is Z, then the value stored in the corresponding counter is *zero.*

Without the loss of generality, we assume that two-counter machines check and modify only one of their counters during any transition, thus, the rule set R contains transition rules of the form $(q, x, c_i) \rightarrow (q', e)$ where $x \in \Sigma \cup \{B\} \cup \{\lambda\}$ corresponds to the symbol scanned on the input tape in state $q \in E$, and $c_i \in \{Z, B\}$, $i \in \{1, 2\}$ correspond to the symbols scanned on the ith storage tape. By a rule of the above form, M enters state $q' \in E$, and the ith counter is modified according to $e \in \{-1, 0, +1\}$. If $x \in \Sigma \cup \{B\}$, then the machine was scanning x on the input tape, and the head moves one cell to the right; if $x = \varepsilon$, then the machine performs the transition irrespective of the scanned input symbol, and the reading head does not move.

A word $w \in \Sigma^*$ is accepted by the two-counter machine if starting in the initial state q_0, the input head reaches and reads the rightmost non-blank symbol on the input tape, and the machine is in the accepting state q_F. Two-counter machines are computationally complete; they are just as powerful as Turing machines.

Theorem 1.

$$\mathcal{L}(PColA, X) = RE, \ \ where\ X \in \{nt, ntmax, ntmin\}.$$

Proof. Let $L \in \Sigma^*$ be an arbitrary recursively enumerable language, and let $M = (\Sigma, Q, q_0, q_f, Tr)$ be a two-counter machine as above with $L = L(M)$.

Let us construct the PCol automaton $\Pi = (V, e, w_E, (w_1, P_1), (w_2, P_2), F)$ where $V = \Sigma \cup Q \cup \{t, t', t'', t''' \mid t \in Tr\} \cup \{c_1, c_2, A\}$, $w_1 = q_0 e$, $w_2 = ee$, $F = \{(\varepsilon; u, q_f e, ee) \mid u \in V^*\}$, and the sets of programs are defined as follows.

For any $\alpha \in \{B, Z\}$, $\beta \in \{-1, 0, +1\}$, we define the disjoint sets of transitions $Tr_{\alpha,\beta} \subseteq Tr$ as follows: $t \in Tr_{\alpha,\beta}$, if and only if, $t : (q, x, i, \alpha) \rightarrow (w, \beta)$, $x \in \Sigma \cup \{\varepsilon\}$, $i \in \{1, 2\}$. Thus, $Tr = Tr_{B,-1} \cup Tr_{B,0} \cup Tr_{B,+1} \cup Tr_{Z,0} \cup Tr_{Z,+1}$.

Now we define the sets of programs as

$$P_1 = \bigcup_{t \in Tr} P_{1,t} \text{ and } P_2 = \bigcup_{t \in Tr} P_{2,t},$$

where for $t \in (Tr_{B,-1} \cup Tr_{B,0})$ we have

$$P_{1,t} = \{p_{t,1} : \langle q \to t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow c_i; r_{t,2} \rangle, p_{t,3} : \langle t' \to t''; c_i \to e \rangle,$$
$$p_{t,4} : \langle t'' \to e; e \leftrightarrow t''' \rangle, p_{t,5} : \langle t''' \to s; e \to e \rangle\},$$

where $r_{t,1}$ and $r_{t,2}$ are the rules $e \overset{T}{\to} a$ and $a \to t'$, respectively, if $t \in Tr$ is such, that $x = a \in \Sigma$, otherwise, if $x = \varepsilon$, then $r_{t,1} = e \to e$ and $r_{t,2} = e \to t'$.

If $t \in Tr_{B,+1}$, then we have

$$P_{1,t} = \{p_{t,1} : \langle q \to t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow c_i; r_{t,2} \rangle, p_{t,3} : \langle t' \to t''; c_i \to c_i \rangle,$$
$$p_{t,4} : \langle t'' \to e; c_i \leftrightarrow t''' \rangle, p_{t,5} : \langle t''' \to s; e \to e \rangle\}$$

with $r_{t,1}$ and $r_{t,2}$ as above.

For these types of transitions, the set P_2 is defined as follows. If $t \in Tr_{B,-1}$, then we have

$$P_{t,2} = \{p_{t,6} : \langle e \leftrightarrow t; e \to e \rangle, p_{t,7} : \langle t \to t'''; e \to e \rangle, p_{t,8} : \langle t''' \leftrightarrow e; e \to e \rangle\},$$

otherwise, if $t \in (Tr_{B,0} \cup Tr_{B,+1})$, then

$$P_{t,2} = \{p_{t,6} : \langle e \leftrightarrow t; e \to c_i \rangle, p_{t,7} : \langle t \to t'''; c_i \leftrightarrow e \rangle\}.$$

Now, if $t \in Tr_{Z,0}$, then we have in P_1

$$P_{t,1} = \{p_{t,1} : \langle q \to t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow e; r_{t,2} \rangle, p_{t,3} : \langle e \leftrightarrow c_i; t \to A \rangle,$$
$$p_{t,4} : \langle t' \to e; e \leftrightarrow t'' \rangle, p_{t,5} : \langle e \leftrightarrow e; t'' \to s \rangle\}$$

where $r_{t,1}$ and $r_{t,2}$ are the rules $e \overset{T}{\to} a$ and $a \to t'$, respectively, if the transition is such, that the input symbol is $x = a \in \Sigma$, otherwise if $x = \varepsilon$, then $r_{t,1} = e \to e$ and $r_{t,2} = e \to t'$.

If $t \in Tr_{Z,+1}$, then

$$P_{t,1} = \{p_{t,1} : \langle q \to t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow e; r_{t,2} \rangle, p_{t,3} : \langle e \leftrightarrow c_i; t \to A \rangle,$$
$$p_{t,4} : \langle t' \to c_i; e \leftrightarrow t'' \rangle, p_{t,5} : \langle c_i \leftrightarrow e; t'' \to s \rangle\}$$

where $r_{t,1}$ and $r_{t,2}$ are as above.

The set P_2 contains only two programs in both cases, these are defined as

$$P_{t,2} = \{p_{t,6} : \langle e \leftrightarrow t; e \to t'' \rangle, p_{t,7} : \langle t \to e; t'' \leftrightarrow e \rangle\}$$

for all $t \in (Tr_{Z,0} \cup Tr_{Z,+1})$.

The PCol automaton Π simulates the work of the two-counter machine M by reading the input symbols with its tape programs and keeping track of the contents of the ith counter as the number of c_i, $i \in \{1, 2\}$ objects present in the environment.

Each transition of M is simulated separately. One of the symbols inside the first cell of Π is from Q, it corresponds to the internal state of M during the simulation process. This symbols is changed through a series of programs into the symbols $s \in Q$ if and only if, M can also change its state from q to s while the counter contents are also checked and modified with an interplay of programs from the two cells of Π.

The reader may check that the PCol automaton Π may reach a final configuration after reading the whole input, if and only if the simulated two-counter machine is able to reach the internal state q_f after processing the same input string using its transitions from Tr.

4 The other computation modes

First we consider the power of the t, $tmax$, and $tmin$ computation modes. Note that a PCol automaton working in these modes reads one input symbol in every computational step, thus, the length of the computation cannot be more than the length of the input string.

As we have seen in Example 2, any regular language can be accepted by a PCol automaton with one cell. Now we present an example showing that the class of languages characterized by PCol automata in the t, $tmax$, or $tmin$ modes contains non-context-free languages.

Example 3. There exists PCol automaton accepting language $L = \{a^n b^n c^n \mid n \geq 0\}$ in any of the computation modes t, $tmax$, or $tmin$. To see this, we construct $\Pi = (\{a, b\}, e, \varepsilon, (w, P), F)$ where $w = ea$,

$$P = \{\langle e \xrightarrow{T} a; a \leftrightarrow e\rangle, \langle a \xrightarrow{T} b; e \leftrightarrow a\rangle, \langle a \xrightarrow{T} b; b \leftrightarrow a\rangle,$$
$$\langle a \xrightarrow{T} c; b \rightarrow b\rangle, \langle b \xrightarrow{T} c; c \leftrightarrow b\rangle\},$$

and $F = \{(\varepsilon; u, cb), (\varepsilon; u, ea) \mid u \in \{c\}^*\}$.

To see how Π works, consider a computation for the input word $aabbcc$.

After the last step, the input tape is read and the automaton is in the final state $(\varepsilon; c, cb)$. It is not difficult to see that Π can only reach a final state if the input is of the form $\{a\}^*\{b\}^*\{c\}^*$ with an equal number of each type of symbols.

Since P contains only tape programs, Π cannot execute any n-transitions, and since it has only one cell, $L(\Pi, t) = L(\Pi, tmax) = L(\Pi, tmin) = L$

step	cell	environment	unread part of tape	applied program
1.	ea	ε	$\mathbf{a}abbcc$	$\langle e \xrightarrow{T} a; a \leftrightarrow e\rangle$
2.	ae	a	$\mathbf{a}bbcc$	$\langle e \xrightarrow{T} a; a \leftrightarrow e\rangle$
3.	ae	aa	$\mathbf{b}bcc$	$\langle a \xrightarrow{T} b; e \leftrightarrow a\rangle$
4.	ba	a	$\mathbf{b}cc$	$\langle a \xrightarrow{T} b; b \leftrightarrow a\rangle$
5.	ba	b	$\mathbf{c}c$	$\langle a \xrightarrow{T} c; b \rightarrow b\rangle$
6.	cb	b	$\mathbf{c}$	$\langle b \xrightarrow{T} c; c \leftrightarrow b\rangle$
7.	cb	c	ε	

Now we consider the initial mode. This time, although the computations can be of arbitrary length, n-transitions can only be executed after the whole input string is processed. The next example demonstrates that the class of languages characterized by PCol automata in the initial mode contains non-semilinear languages.

Example 4. There exists a PCol automaton Π, such that $L(\Pi, init) = \{a^{2^n}\}$. To see this, consider the PCol automaton $\Pi = (V, e, \varepsilon, (ee, P_1), (ee, P_2), F)$ where $V = \{a, b, B, c, c', d, d', f, f', g, g', i, i', \underline{i}, \underline{\underline{i}}, \underline{i}', \underline{\underline{i}}', x, x', x'', x''', \overline{x}, \overline{\overline{x}}, x_h, y, y', y'', z, u, u', u'', u''', \overline{u}, \overline{\overline{u}}, v, v', v''\}$, $F = \{(\varepsilon; \varepsilon, x_h e, ee)\}$, and

$$P_1 = P_{1,in} \cup P_{1,div} \cup P_{1,b} \cup P_{1,B} \cup P_{1,tran} \cup P_{1,fin}, \text{ and } P_2 = P_{2,b} \cup P_{2,B}$$

where the set of programs are defined as follows.

$$P_{1,in} = \{p_1 : \langle e \xrightarrow{T} a; e \rightarrow b\rangle, p_2 : \langle a \xrightarrow{T} a; b \leftrightarrow e\rangle, p_3 : \langle a \xrightarrow{T} a; e \rightarrow b\rangle\}.$$

Using these programs, the first cell reads the input symbols and puts one object b into the environment after reading two as.

After reading the input, the cells may replace two bs by one B, or two Bs by one b. This is achieved by the programs:

$$\begin{aligned} P_{1,div} = \{&p_4 : \langle a \rightarrow c; e \leftrightarrow b\rangle, p_5 : \langle c \rightarrow d; b \rightarrow e\rangle, p_6 : \langle d \rightarrow f; e \leftrightarrow b\rangle,\\ &p_7 : \langle f \rightarrow g; b \rightarrow B\rangle, p_8 : \langle g \rightarrow i; B \leftrightarrow e\rangle, p_9 : \langle i \rightarrow c; e \leftrightarrow b\rangle,\\ &p_{10} : \langle i' \rightarrow c'; e \leftrightarrow B\rangle, p_{11} : \langle c' \rightarrow d'; B \rightarrow e\rangle,\\ &p_{12} : \langle d' \rightarrow f'; e \leftrightarrow B\rangle, p_{13} : \langle f' \rightarrow g'; B \rightarrow b\rangle,\\ &p_{14} : \langle g' \rightarrow i'; b \leftrightarrow e\rangle\}. \end{aligned}$$

After exchanging the bs to Bs or reversely, the cells have to control if there is any remaining bs or Bs and only in the negative case an the computation continue. This is done by the interplay of the programs

$$\begin{aligned} P_{1,b} = \{&p_{15} : \langle e \rightarrow x'; x \leftrightarrow e\rangle, p_{16} : \langle e \rightarrow x''; x' \leftrightarrow e\rangle p_{17} : \langle x'' \rightarrow x'''; e \leftrightarrow e\rangle,\\ &p_{18} : \langle x''' \rightarrow \overline{x}; e \leftrightarrow e\rangle, p_{19} : \langle \overline{x} \rightarrow \overline{\overline{x}}; e \leftrightarrow e\rangle, p_{20} : \langle \overline{\overline{x}} \rightarrow \underline{i}; e \leftrightarrow y''\rangle,\\ &p_{21} : \langle \overline{\overline{x}} \rightarrow \underline{i}'; e \leftrightarrow y\rangle, p_{22} : \langle y'' \rightarrow i; \underline{i} \leftrightarrow e\rangle, p_{23} : \langle y \rightarrow z; \underline{i}' \leftrightarrow e\rangle,\\ &p_{24} : \langle e \rightarrow \underline{\underline{i}}'; z \leftrightarrow \underline{i}'\rangle, p_{25} : \langle \underline{i}' \rightarrow i'; \underline{\underline{i}}' \leftrightarrow e\rangle\}, \end{aligned}$$

$$\begin{aligned}P_{2,b} = \{&p_{26} : \langle e \to y; e \leftrightarrow x\rangle, p_{27} : \langle x \to y'; y \leftrightarrow x'\rangle, p_{28} : \langle x' \to y''; y' \leftrightarrow b\rangle,\\ &p_{29} : \langle b \to b; y'' \leftrightarrow y\rangle, p_{30} : \langle y \to e; b \leftrightarrow e\rangle, p_{31} : \langle x' \to e; y' \leftrightarrow z\rangle,\\ &p_{32} : \langle z \to e; e \leftrightarrow e\rangle\},\end{aligned}$$

for checking bs, and the programs

$$\begin{aligned}P_{1,B} = \{&p_{33} : \langle e \to u'; u \leftrightarrow e\rangle, p_{34} : \langle e \to u''; u' \leftrightarrow e\rangle, p_{35} : \langle u'' \to u'''; e \leftrightarrow e\rangle,\\ &p_{36} : \langle u''' \to \overline{u}; e \leftrightarrow e\rangle, p_{37} : \langle \overline{u} \to \overline{\overline{u}}; e \leftrightarrow e\rangle, p_{38} : \langle \overline{\overline{u}} \to \underline{i'}; e \leftrightarrow v''\rangle,\\ &p_{39} : \langle \overline{\overline{u}} \to \underline{i}; e \leftrightarrow v\rangle, p_{40} : \langle v'' \to i'; \underline{i'} \leftrightarrow e\rangle, p_{41} : \langle v \to w; \underline{i} \leftrightarrow e\rangle,\\ &p_{42} : \langle e \to \underline{\underline{i}}; w \leftrightarrow \underline{i}\rangle, p_{43} : \langle \underline{i} \to i; \underline{\underline{i}} \leftrightarrow e\rangle\},\end{aligned}$$

$$\begin{aligned}P_2 = \{&p_{44} : \langle e \to v; e \leftrightarrow u\rangle, p_{45} : \langle u \to v'; v \leftrightarrow u'\rangle, p_{46} : \langle u' \to v''; v' \leftrightarrow B\rangle,\\ &p_{47} : \langle B \to B; v'' \leftrightarrow v\rangle, p_{48} : \langle v \to e; B \leftrightarrow e\rangle, p_{49} : \langle u' \to e; v' \leftrightarrow w\rangle,\\ &p_{50} : \langle w \to e; e \leftrightarrow e\rangle\},\end{aligned}$$

for checking Bs.

The following programs are used for connecting the different phases of the functioning of the system,

$$P_{1,tran} = \{p_{51} : \langle i \to x; e \leftrightarrow e\rangle, p_{52} : \langle i' \to u; e \leftrightarrow e\rangle\},$$

and for finishing the computation

$$P_{1,fin} = \{p_{53} : \langle d \to x_h; e \leftrightarrow e\rangle, p_{54} : \langle d' \to x_h; e \leftrightarrow e\rangle, p_{55} : \langle a \to x_h; b \to e\rangle\}.$$

5 Conclusion

P colony automata are very simple language recognizing devices, with strong formal resemblance to finite automata. Especially interesting are those cases when the function of these constructs is governed by the use of their tape rules, i.e., the computational modes t, $tmin$ and $tmax$. The description of the exact computational power of these variants of PCol automata is a challenging problem. We guess to obtain language classes of very low complexity.

References

1. L. Ciencialová, L. Cienciala, Variation on the theme: P colonies. In: Proc. 1st Intern. Workshop on Formal Models. (D. Kolăr, 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. Cienciala, L. Ciencialová, A. Kelemenová, Homogeneous P colonies. Computing and Informatics 27 (2008), 481–496.
4. 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.
5. 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.
6. 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.
7. E. Csuhaj-Varjú, M. Oswald, Gy. Vaszil, P automata. Chapter 6, In: The Oxford Handbook of Membrane Computing. (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2010, 144–167.
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), 569–502.
11. J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multi-agent systems. Cybernetics and Systems 23 (1992), 621–633.
12. 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.
13. 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.
14. M. Minsky, Computation – Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967.
15. Gh. Păun, Membrane Computing – An Introduction. Springer-Verlag, Berlin, 2002.
16. The Oxford Handbook of Membrane Computing. (Gh. Păun, G. Rozenberg, A. Salomaa, eds.) Oxford University Press, 2010.

A Cellular Sudoku Solver

Daniel Díaz-Pernil, Carlos M. Fernández-Márquez, Manuel García-Quismondo, Miguel A. Gutiérrez-Naranjo, Miguel A. Martínez-del-Amor

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
`sbdani@us.es, carfermar@alum.us.es, mangarfer2@alum.us.es, magutier@us.es, mdelamor@us.es`

Summary. Sudoku is a very popular puzzle which consists on placing several numbers in a squared grid according to some simple rules. In this paper we present an efficient family of P systems which solve sudoku puzzles of any order verifying a specific property. The solution is searched by using a simple human-style method. If the sudoku cannot be solved by using this strategy, the P system detects this drawback and then the computations stops and returns `No`. Otherwise, the P system encodes the solution and returns `Yes` in the last computation step.

1 Introduction

Sudoku is currently one of the most famous puzzles in the world. The most popular version consists on a 9×9 grid made up of 3×3 subgrids, but the general case, an $n^2 \times n^2$ grid with $n \times n$ subgrids is considered. Some cells contain numbers, which can be considered as *input data*. The goal is to fill in the empty cells, one number in each, so that each column, row, and subgrid contains the numbers 1 through 9 exactly once (numbers 1 to n^2 in the general case). If the input data are correct, the sudoku has one and only one solution.

The creator is believed to be Howard Garns [6]. He is likely to be the inventor of a puzzle called "Number Place" that appeared in New York in 1979. The puzzle was introduced in Japan by the publishing company Nikoli in the paper *Monthly Nikolist* in April 1984 as *Suji wa dokushin ni kagiru* [7], which can be translated as *the numbers must occur only once* or *the numbers must be single*. Later, the name was abbreviated as *sudoku*, where *su* stands for *number* and *doku* stands for *alone*. Later Wayne Gould from New Zealand discovered the puzzle on a trip to Japan and wrote a program to generate new puzzles. He convinced The Times of London to publish Sudoku puzzles in 2004.

In addition to its undoubted success in entertainment, sudoku has important properties from a mathematical point of view. The first natural question is to wonder about is the number of all possible sudoku grids. The answer to this question is not an easy matter. A valid sudoku solution is also a Latin square. A Latin square is an $n \times n$ table filled by using numbers from 1 to n in such way that each symbol occurs exactly once in each row and exactly once in each column. The number of 9×9 Latin squares is about 5.525×10^{27}.

Sudoku imposes the additional constraint on subgrids, so from the previous number we need to remove the Latin squares which do not satisfy the condition. The number of valid sudoku solution grids for the standard 9×9 grid is 6,670,903,752,021,072,936,960. This number is equal to $9! \times 72^2 \times 2^7 \times 27,704,267,971$, the last factor of which is prime. The result was derived through logic and brute force computation. The details can be found at [1]. Other important property is that it has been proved that the general problem of solving sudoku puzzles on $n^2 \times n^2$ grids of $n \times n$ boxes is known to be **NP**-complete [5].

Nonetheless, the number of possible solution grids is not the object of study of this paper nor the complexity of finding the solution. In this paper we study the problem of solving sudoku by using Membrane Computing techniques. In the first part of the paper, we develop a theoretical study about the use of brute force algorithms to solve it, based on a well-known solution for the SAT problem. As we will see below, the number of elementary membranes for a usual 9×9 sudoku exceeds the number of atoms of the observable universe, so we have a good reason for looking for a different strategy.

In the second part of the paper, we present a family of P systems $\{\Pi(n)\}_{n \in \mathbb{N}}$ such that $\Pi(n)$ is a P system with input. Such an input is, of course, the input for one sudoku puzzle encoded as a multiset. The solution is searched by using a human-style method based on looking for squares where only one candidate can be placed. This method is good enough to find the solution for a large amount of sudokus, but not all the sudokus can be solved by using this method. An original control method in the design of the algorithm is that the P system stops if the sudoku cannot be solved, i.e., instead of going into a non-ending search, the P system detects the drawback and halts. If the solution can be reached, the P systems stops, sends out an object `Yes` to the environment and provides the solution encoded on the skin. Otherwise, if the P system detects that the solution cannot be reached then it halts and sends `No` to the environment in the last step of computation.

The paper is organized as follows: first we explore a theoretical brute force algorithm based on a Membrane Computing solution for the SAT problem. After showing the practical drawback of such a solution, we present our efficient family of P systems for solving a large amount of sudokus. We illustrate the behavior of this family with an overview of the computation and, finally, some final remarks are presented.

2 A Brute Force Algorithm

A *sudoku square* of order n consists of n^4 constants (usually n^2 copies of the numbers $1, 2, \ldots, n^2$), arranged into an $n^2 \times n^2$ grid which comprises n^2 subgrids of size $n \times n$ (also called boxes). Such a grid verifies that the entries in each row, each column and each box are all different. A *sudoku problem* consists on a partial assignment of the variables in a Sudoku square. The target is to find a completion of the assignment which extends the partial assignment and satisfies the constraints.

The first idea for solving a sudoku problem is to consider it as a constraint problem. In fact, we are looking for one assignment of numbers to squares which satisfies a finite amount of restrictions. The set of constraints of a sudoku problem can be expressed as a logic formula in conjunctive normal form. Following [3], a sudoku square of order n can be represented as an instance of the SAT problem with n^6 propositional variables. For each entry in the $n^2 \times n^2$ grid, we will consider n^2 variables. Let us use the notation s_{xyz} to refer to variables. Variable s_{xyz} is assigned *true* if and only if the entry in the row x and column y is number z. In this way, if the variable s_{753} takes the value *true*, then it means that the number 3 is placed at position $(7, 5)$ of the grid. According to this notation, the different constraints for the sudoku problem can be represented as the following formulae:

- There is at least one number in each entry:
$$\psi_1 \equiv \bigwedge_{x=1}^{n^2} \bigwedge_{y=1}^{n^2} \bigvee_{z=1}^{n^2} s_{xyz}$$
- Each number appears at most once in each row:
$$\psi_2 \equiv \bigwedge_{y=1}^{n^2} \bigwedge_{z=1}^{n^2} \bigwedge_{x=1}^{n^2-1} \bigwedge_{i=x+1}^{n^2} (\neg s_{xyz} \vee \neg s_{iyz})$$
- Each number appears at most once in each column:
$$\psi_3 \equiv \bigwedge_{x=1}^{n^2} \bigwedge_{z=1}^{n^2} \bigwedge_{y=1}^{n^2-1} \bigwedge_{i=y+1}^{n^2} (\neg s_{xyz} \vee \neg s_{xiz})$$
- Each number appears at most one in each box:
$$\psi_4 \equiv \bigwedge_{z=1}^{n^2} \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \bigwedge_{x=1}^{n} \bigwedge_{y=1}^{n} \bigwedge_{k=y+1}^{n} (\neg s_{(ni+x)(nj+y)z} \vee \neg s_{(ni+x)(nj+k)z})$$
$$\psi_5 \equiv \bigwedge_{z=1}^{n^2} \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \bigwedge_{x=1}^{n} \bigwedge_{y=1}^{n} \bigwedge_{k=x+1}^{n} \bigwedge_{l=1}^{n} (\neg s_{(ni+x)(nj+y)z} \vee \neg s_{(ni+k)(nj+l)z})$$

The conjunction of these five formulae $\Phi \equiv \psi_1 \bigwedge \psi_2 \bigwedge \psi_3 \bigwedge \psi_4 \bigwedge \psi_5$ is a formula in conjunctive normal form and each truth assignment which makes it true represents a right arrangement of n^2 copies of $1, 2, \ldots, n^2$ according to the sudoku constraints.

Once expressed the sudoku as such a formula, finding a solution to the puzzle can be considered as the problem of finding a truth assignment which satisfies the formula. This is exactly the satisfiability (SAT) problem.

Any truth assignment which makes it true represents a right arrangement of n^2 copies of $1, 2, \ldots, n^2$ according to the sudoku constraints over an empty sudoku grid of order n, but we are not interested in finding such solutions. In fact, a sudoku puzzle should have a certain amount of numbers placed in the right position as input in such way that there exists a unique possible assignment which represents the solution to the problem. Given a sudoku puzzle, we will call *Input* to the set

$$Input = \{\langle x, y, z\rangle \,:\, z \text{ is placed on the square } (x, y) \text{ of the grid}\}$$

In order to deal with the input, it is enough to add the corresponding values to the formula Φ as follows:

$$\Phi_{in} \equiv \Phi \wedge \bigwedge_{\langle x,y,z\rangle)\in Input} s_{xyz}$$

Given a sudoku problem and its associated formula Φ_{in}, finding the solution to the problem is equivalent to finding a truth assignment which satisfies the formula.

In [2], a family of P systems with active membranes to solve the SAT problem was presented. It was based on the solution presented in [4]. The main difference was that the solution from [4] only provides `Yes` or `No` as answers to the problem. The solution in [2] found and stored all the truth assignments which satisfy the formula, if there exists.

By considering the encoding of a sudoku problem as a CNF formula on one side and the family of P systems which provides solutions for SAT on the other side, we have a Membrane Computing solution for all sudoku problems.

This nice theoretical solution has an insurmountable obstacle from a practical point of view: The number of elementary membranes in one configuration of one P system from the family reaches 2^N, where N is the number of variables of the CNF formula. For a sudoku of order n, the number of variables is n^6, and so the number of elementary membranes is 2^{n^6}. For a usual sudoku of order 3, 2^{729} elementary membranes are simultaneously handled. Estimates of the matter content of the observable universe indicate that it contains on the order of 10^{80} atoms[1], so the brute force algorithm is only a fine calculus for Membrane Computing theory.

3 A New Solution

In this section we present a family of P systems $\mathbf{P} = \{\Pi(n) \,:\, n \in \mathbb{N}\}$ such that $\Pi(n)$ solves sudokus of order n. The P system $\Pi(n)$ receives as input the initial data placed in a sudoku puzzle. It is designed to solve sudokus which satisfy a property which will be described below. If the sudoku satisfies the property, the

[1] http://en.wikipedia.org/wiki/Observable_universe

P system computes the solution, it sends an object `Yes` to the environment in the last step of computation and encodes the solution to the sudoku as a multiset in the skin of the halting configuration. Otherwise, the P system detects that the property is not satisfied and halts by sending an object `No` to the environment in the last step of computation.

The property is the following: *In all partial solutions of the sudoku, there exists at least one square* (i, j) *with a unique candidate.*

We will call *partial solution* of a sudoku to a sudoku grid where some new numbers have been placed and all of them are in the right position. A number p is a unique candidate for the square (i, j) if for all $q \in \{1, \ldots, n^2\}$, $p \neq q$, q has been previously placed in the same row, the same column or the same box of (i, j). For example, if we consider the sudoku of order 2 of Figure 1, number 4 is a unique candidate for the square $(1, 2)$, since 1 is in the same row, 2 is in the same column and 3 is in the same box.

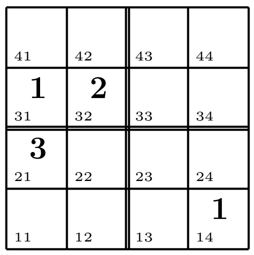


Fig. 1. A sudoku problem of order 2

Many sudokus satisfy this property. It is in the base of many human strategies for solving sudokus. Nonetheless, sometimes it is not enough to solve the sudoku and more sophisticated methods are necessary.

3.1 A Family of P Systems

Next, we present a family $\mathbf{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ for solving any sudoku of order n verifying the property stated above. Each P system $\Pi(n)$ only depends only on the order n of the sudoku and it does not increase the number of membranes along the computation. The used rules are of the following types:

- *Enzymatic rules:* $[\neg in\ u \underset{cat}{\longrightarrow} v]_e$. The multiset u evolves to the multiset v in the membrane with label e. The rule is applied if in the same membrane the objects from the set cat are present (catalysts) and none of the objects from the set in are present (inhibitors). The catalysts and the inhibitor are not modified by the application of the rules and cat, in and v can be empty.
- *Dissolution rules:* $[\,u\,]_e \rightarrow o$. The multiset u causes membrane e to dissolve and produces the object o.

- *Send-out rules:* $[\,a\,]_e \rightarrow [\,]_e\, a$. The object a is sent out of the membrane with label e.

As usual, 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, should evolve. If a membrane is dissolved, its content is left free in the surrounding region. If there are objects in this membrane which evolve by means of enzymatic rules and a membrane h is dissolved at the same time, then we suppose that first the enzymatic rules are used and then the dissolution is produced. Of course, this process takes only one step. We will also use priorities among sets of rules.

The input will be provided as a set of objects z_{ijn} by denoting that the number n is placed at the square with row i and column j. The initial configuration will also contain information about the box corresponding to each square. In such a way, objects box_{ijk} with $i, j \in \{1, \ldots, n^2\}$ are place in the initial configuration and k is the box corresponding to the square (i, j), i.e., if $i = \alpha n + \beta$ and $j = \gamma n + \delta$ with $\alpha, \gamma \in \{0, \ldots, n-1\}$ and $\beta, \delta \in \{1, \ldots, n\}$ then $k = \alpha n + \gamma + 1$.

The initial configuration also contains the objects f_{ix}, c_{jx}, b_{kx}, sq_{ij} with $i, j, k, x \in \{1, \ldots, n^2\}$. The occurrence of f_{ix} in the configuration denotes that the number x is not placed in any square of the row i yet and then, the number x can be eventually placed in such a row in the future. Analogously, c_{jx} denotes that x is not placed in the column j and b_{kx} that the object is not placed in any square of the box k. The objects sq_{ij} are witnesses of the existence of the corresponding square. Finally, n^2 copies of each object r_{ij} with $i, j \in \{1, \ldots, n^2\}$ are also placed in the initial configuration.

The idea of the design is to develop a sequence of two stages: The *checking stage* and the *reset stage*. In the checking stage, the P system looks for squares with a unique candidate. If such squares are found, the candidates are placed in them. After the stage all the auxiliary objects are recalculated in the *reset stage* and then we start again the *checking stage*. This *checking-reset* cycle ends when all the squares are filled and the sudoku is solved or if in a checking stage no new squares with unique candidates are found.

Formally, the P system of order n with input that solves the sudokus with the property claimed above is a construct

$$\Pi(n) = \langle \Gamma, H, \mu, w_e, w_s, i_0, R_1, R_2, R_3^a, R_3^m, R_3^d, R_4, \ldots, R_{11}, R_{12}^1, R_{12}^2 \rangle$$

with the priorities $R_1 > R_2 > R_3^q > R_4 > \ldots R_{11} > R_{12}^x$ with $q \in \{a, r, m\}$ and $x \in \{1, 2\}$ where

- The alphabet $\Gamma = \{s_{ijx}, z_{ijx}, box_{ijk}, sq_{ij}, f_{ix}, c_{jx}, b_{kx}, a_{ij}, r_{ij} \;:\; i, j, k, x \in \{1, \ldots, n^2\}\} \cup \{k_0, k_1, w\}$
- The set of labels $H = \{e, s\}$
- The membrane structure $\mu = [\,[\,]_e\,]_s$
- The initial multisets $w_e = \{k_1\} \cup \{box_{ijk}, sq_{ij}, f_{ix}, c_{jx}, b_{kx}, r_{ij}^{n^2} \;:\; i, j, k, x \in \{1, \ldots, n^2\}\}$ and $w_s = \emptyset$.

- $i_0 = e$, i.e., the input membrane is e.

We also consider the following sets of rules[2]

$$R_1 \equiv [\, z_{ijx} \to s_{ijx}\, p\,]_e \text{ for } i,j,x \in \{1,\dots,n^2\}.$$

Each input object z_{ijx} produces an object s_{ijx} and one object p. In any configuration we will have as many objects p as numbers are correctly placed on the sudoku. After applying these rules, we have as many objects p as numbers are placed as input.

$$R_2 \equiv \left\{ \begin{array}{l} [f_{ix} \xrightarrow[s_{ijx}]{} \lambda]_e \\ [c_{jx} \xrightarrow[s_{ijx}]{} \lambda]_e \\ [b_{kx} \xrightarrow[s_{ijx}\, box_{ijk}]{} \lambda]_e \end{array} \right\} \text{ for } i,j,k,x \in \{1,\dots,n^2\}.$$

The object s_{ijx} represents that the number x is placed in the square (i,j). When such an object is generated the objects f_{ix}, c_{jx} and b_{kx} must disappear.

$$\left. \begin{array}{l} R_3^m \equiv [\neg d\, m_{ijx} \xrightarrow[k_1]{} \lambda]_e \\ R_3^a \equiv [\neg d\, a_{ij} \xrightarrow[k_1]{} \lambda]_e \\ R_3^r \equiv [\neg d\, r_{ij} \xrightarrow[k_1]{} \lambda]_e \end{array} \right\} \text{ for } i,j,x \in \{1,\dots,n^2\}.$$

$$R_4 \equiv [\neg d\, \neg r_{ij} sq_{ij} \xrightarrow[k_1]{} sq_{ij}\, r_{ij}^{n^2}\, d]_e \text{ for } i,j \in \{1,\dots,n^2\}.$$

Before starting with the checking stage, we ensure that the markers (m_{ijx}) and counters (a_{ij} and r_{ij}) are reset. First, we remove all the copies of a_{ij}, m_{ij} and r_{ij}. In the next step we add n^2 copies of each object r_{ij} to membrane e.

$$R_5 \equiv [k_1 \to k_0]_e$$

The reset stage ends when object k_1 (used as catalyst in the previous sets of rules) evolves to k_0.

$$R_6 \equiv [\neg m_{ijx}\, r_{ij} \xrightarrow[f_{ix}\, c_{jx}\, b_{kx}\, box_{ijk}]{} m_{ijx}\, a_{ij}]_e \text{ for } i,j,x \in \{1,\dots,n^2\}.$$

The checking stage starts with the set R_6. We know that if objects $f_{ix}\, c_{jx}\, b_{kx}$ are present in membrane e in one configuration, then the number x is a candidate to be placed in the square (i,j), since x has not been placed yet in the row i, the column j or the box k. The question is to know if x is the *unique* candidate. This is checked by rules from set R_6. Before applying these rules, we have checked that for all square (i,j) we have n^2 copies of r_{ij} in membrane e and zero copies of a_{ij}. If $f_{ix}\, c_{jx}\, b_{kx}$ are present in the membrane (they act as catalyst), then the

[2] We write $\neg$ before the object a if a acts as an inhibitor.

corresponding rule is applied. The application of the rule removes one copy of r_{ij} and produces one copy of a_{ij}. The occurrence of m_{ijx} ensures that the rule is applied once.

$$R_7 \equiv [\neg s_{ijq}\, r_{ij}^{n^2-1}\, a_{ij} f_{ix}\, c_{jx}\, b_{kx} \xrightarrow[d\, box_{ijk}]{} s_{ijx}\, p\, w]_e \; i,j,x,q \in \{1,\dots,n^2\}.$$

If there exists only one a_{ij} and n^2-1 copies of r_{ij} (and the square (i,j) is empty) then the square (i,j) has a unique candidate. The rules delete the objects $f_{ix}\, c_{jx}\, b_{kx}$ and introduce the objects s_{ijx}, p and w. The object s_{ijx} corresponds to a number and a square in the solution of the sudoku, p denotes that a new number has been placed in the solution (when p reaches n^2 copies, the cycle *reset-checking* stops) and w is a witness of the application of the rule.

$$R_8 \equiv [w\, k_0 \to k_2]_e$$

$$R_9 \equiv \begin{cases} [w \to \lambda]_e \\ [k_0]_e \to \texttt{No} \end{cases}$$

If an object w has been produced, it means that at least one of the rules from the set R_7 has been applied. In other words, a new number has been placed on the solution of the sudoku and the *reset-checking* cycle must go on. In this case, objects w and k_0 are consumed by the rule from R_8 and a new object k_0 is produced. If k_0 has not been consumed by the rule from R_8, then no new number has been placed on the sudoku. This means that it does not make sense going on with the *reset-checking* cycle, since the next checking stage will have the same configuration that this one. In order to prevent an infinite sequence of cycles, object k_0 dissolves membrane e and the remaining objects w (if any) are removed.

$$R_{10} \equiv [p^{n^6}]_e \to \texttt{Yes}$$

The copies of p denote the number of squares correctly filled. When it reach n^6 copies, the membrane e is dissolved.

$$R_{11} \equiv \begin{cases} [d \to \lambda]_e \\ [k_2 \to k_1]_e \end{cases}$$

This set of rules marks the end of the checking stage. The inhibitor d is removed and the catalyst k_1 is produced so rules from R_3 can be triggered and the reset stage starts again.

Notice that membrane e is dissolved anyway. If the sudoku is not completed, but no more numbers can be placed, then the rule $[k_0]_e \to \texttt{No}$ is applied and the object No appears in membrane s. Otherwise, if the sudoku is completed, $[p^{n^6}]_e \to \texttt{Yes}$ is applied and Yes appears in membrane s. In the last step of computation, the corresponding object Yes or No is sent out to the environment.

$$R^1_{12} \equiv \begin{cases} [\texttt{Yes}]_s \to [\,]_s\,\texttt{Yes} \\ [\texttt{No}]_s \to [\,]_s\,\texttt{No} \end{cases}$$

Also in the last step, the auxiliary objects are removed from membrane s

$$R^2_{12} \equiv \begin{cases} [p \to \lambda]_s & [m_{ijx} \to \lambda]_s & [a_{ij} \to \lambda]_s \\ [f_{ix} \to \lambda]_s & [d \to \lambda]_s & [r_{ij} \to \lambda]_s \\ [c_{jx} \to \lambda]_s & [box_{ijk} \to \lambda]_s & [k_2 \to \lambda]_s \\ [b_{kx} \to \lambda]_s & [sq_{ij} \to \lambda]_s & \end{cases}$$

4 An Overview of the Computation

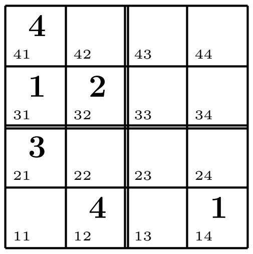
We will give some hints on the computation by following the computation of the example of order two showed in Figure 1. We start with a sudoku problem with 4 numbers placed. Such an input is encoded in the multiset $Input = z_{311}\,z_{322}\,z_{213}\,z_{141}$. The initial configuration C_0 has two membranes $[\,[\,]_e\,]_s$. Membrane s is empty and membrane e contains the input plus the objects k_1 and $f_{ix}\,c_{jx}\,b_{kx}\,sq_{ij}\,box_{ijk}, r^4_{ij}$ for $i, j, k, x \in \{1, \dots, 4\}$.

From this configuration C_0, rules from sets R_1, R^r_3 , R_5 and R_6 can be applied. Due to priority, rules from R_1 are applied and we obtain configuration C_1. The objects z_{ijx} are removed and replaced by the corresponding s_{ijx}. Four copies of object p also appear. This means that four numbers are correctly placed in the sudoku.

Rules from R_1 will nor be applied any more, since no rule produces objects z_{ijx}. From C_1 rules of set R_2 are applied and the objects $f_{31}\,c_{11}\,b_{31}\,f_{32}\,c_{22}\,b_{32}\,f_{23}\,c_{13}\,b_{13}\,f_{11}\,c_{41}\,b_{21}$ are removed from membrane e and we obtain configuration C_2.

Next, rules from R^r_3 are applied (obtaining C_3) and then we apply rules from R_4 (obtaining C_4). This is the first time that we reset the system, so C_4 is identical to C_2, but with new objects d.

Since membrane e contains objects d, none of the rules from sets R^*_3 nor from R_4 are applicable. The next application of the rule corresponds to R_5. The object k_1 evolves to k_0 and the checking stage begins. The new configuration C_5 contains k_0. Since no object m_{ijx} has been created yet, for each triplet (i, j, x) such that the number x can be placed in the square (i, j), one of the rules from R_6 is triggered, reaching configuration C_6. The application of one of these rules removes one copy of the corresponding r_{ij} and produces one copy of a_{ij}. Each rule only can be triggered once due to object m_{ijx}. After the application of these rules, the number of objects a_{ij} denotes the number of possible candidates to be placed in the square (i, j). In the next step, two rules from R_7 are triggered and the objects s_{124} and s_{414} are produced and configuration C_7 is reached. After the application of rules from R_6, the number of objects a_{ij} for squares $(1, 2)$ and $(4, 1)$ was exactly one. This means that for these squares, the candidate is unique. The current partial solution for the sudoku is shown in Figure 2.

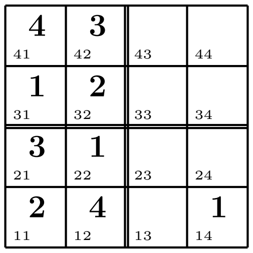


4 41	42	43	44
1 31	**2** 32	33	34
3 21	22	23	24
11	**4** 12	13	**1** 14

Fig. 2. Partial solution at configuration C_7

The application of rules from R_7 has produced two copies of the object w in the configuration C_8. One of these copies, along with k_0 produces k_1 (Configuration C_9). In the next step the remaining w is deleted (Configuration C_{10}). The checking stage finishes with the application of the rule from R_{10}. Object d is removed and the configuration C_{11} is reached.

The object d is a strong inhibitor. Since it has disappeared, rules from R_3 can be applied and the reset stage starts again. We go on with this new reset stage and the application of three rules from R_7 produces the objects s_{221}, s_{423} and s_{112}. This means that three new numbers can be placed on the sudoku (see Figure 3).



4 41	**3** 42	43	44
1 31	**2** 32	33	34
3 21	**1** 22	23	24
2 11	**4** 12	13	**1** 14

Fig. 3. Partial solution at configuration C_{16}

The *checking-reset* cycle goes on and in C_{24}, the objects s_{133} and s_{442} are added. In C_{32}, s_{431}, s_{334} and s_{244} are added, and finally in C_{40}, the objects s_{232} and s_{343} are generated. Figure 4 shows the solution of the sudoku according to the objects s_{ijx} in membrane e.

The P system follows with the computation and in C_{42} there are 16 copies of objects p in membrane e. The rule from R_{10} is triggered, the membrane e is dissolved and all the objects go to membrane s. In the next step, one objects `Yes` is sent out and the remaining objects (but s_{ijx}) are deleted, so the final configuration has an object `Yes` in the environment and one membrane where the solution is encoded.

4 41	**3** 42	**1** 43	**2** 44
1 31	**2** 32	**4** 33	**3** 34
3 21	**1** 22	**2** 23	**4** 24
2 11	**4** 12	**3** 13	**1** 14

Fig. 4. Solution at configuration C_{40}

As pointed above, it is possible that for some sudokus there no exist squares with unique candidates. In such cases, when the checking stage is reached, no rule from R_7 is applied and no object w is produced. Since we do not have objects w, the rule from R_8 is not applied, and according to priority, the dissolution rule from R_9 is applied. An object `No` is sent to membrane s, the reset-checking cycle is stopped. In the next step an object `No` is sent out and the computation ends.

Notice that we have chosen an example of order 2 for illustrating the process, but the computation is similar for a sudoku problem of any order.

5 Final Remarks

P systems have showed many times to be versatile enough to represent many different situations, from real life or from more abstract scenarios. In parallel with a very expressive representation system, Membrane Computing also provides a friendly set of tools for dealing with the information. Besides the computational power of the expressiveness, the research in a new computational model also needs to face a problem of efficiency. In this paper we provide a first theoretical solution for the problem of finding a solution to a sudoku problem. It is based on an appropriate representation of the problem as a formula and a well-established solution of the SAT problem. The solution works from a theoretical point of view, but it does not make sense form a practical one.

In the second part of the paper, we have designed a solution for solving sudoku problems in an effective way. It solves all the sudokus which verifies a very common property, by following a strategy close to human-style solutions. One of the main original contributions of the design is the checking to avoid infinite loops. The implemented strategy is enough to find the solution to many sudoku problems, but it is not enough to solve all of them. The next step is to go on with the research and tackle these *hard* sudoku problems. Beyond these efforts it is the horizon of a better understanding of the cellular processes and making more and more efficient cellular designs.

Acknowledgements

The authors acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. B. Felhenhauer, F. Jarvis. Enumerating possible sudoku grids. Univ. Sheffield and Univ. Dresden, 2005. `http://www.shef.ac.uk/~pm1afj/sudoku/sudoku.pdf`
2. M.A. Gutiérrez–Naranjo, M.A. Martínez–del–Amor, I. Pérez–Hurtado, M.J. Pérez–Jiménez. Solving the $N - Queens$ Puzzle with P systems. In Seventh Brainstorming Week on Membrane Computing, Vol. I, R. Gutiérrez-Escudero, M.A. Gutiérrez-Naranjo, Gh. Păun, Ignacio Pérez-Hurtado, A. Riscos Núñez, (eds.) Fénix Editora (2009) 199-210.
3. I. Lynce, J.Ouaknine. Sudoku as a SAT Problem. Proceedings of the 9 th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale (2006). `http://anytime.cs.umass.edu/aimath06/proceedings/P34.pdf`
4. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrinini. A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil (Eds.). Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, Computer and Automaton Research Institute of the Hungarian Academy of Sciences (2003) 284-294.
5. T. Yato and T. Seta. Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. IEICE Trans. Fundamentals, E86-A (5) (2003) 1052–1060.
6. `http://www.dellmagazines.com`
7. `http://www.nikoli.co.jp/puzzles/`

A Cellular Way to Obtain Homology Groups in Binary 2D Images

Daniel Díaz-Pernil[1], Miguel A. Gutiérrez-Naranjo[2],
Pedro Real[1], Vanesa Sánchez-Canales[1]

[1] Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics
University of Sevilla
`{sbdani,real,vscanales}@us.es`

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
`magutier@us.es`

Summary. In this paper we present a P systems-based solution for the *Homology Groups of Binary 2D Image (HGB2I) Problem*, a classical problem in Homology Theory. To this aim, we present a family of P systems which solves all the instances of the problem in the framework of *Tissue-like P systems with catalysts*. This new framework combines the membrane structure and symport-antiport communication rules of tissue-like P systems with the power of catalysts and inhibitors.

1 Introduction

Homology theory is a branch of Algebraic Topology that attempts to distinguish between spaces by constructing algebraic invariants that reflect the connectivity properties of the space. The field has its origins in the work of the French mathematician, theoretical physicist, and a philosopher of science Jules Henri Poincaré. Homology groups (related to the different n-dimensional holes, connected components, tunnels, cavities, etc., of a geometric object) are invariants from Algebraic Topology which are frequently used in Digital Image Analysis and Structural Pattern Recognition. In some sense, they reflects the topological nature of the object in terms of the number and characteristics of its holes.

In this paper we explore one of the main problems from Homology Theory in terms of Membrane Computing[3]. The chosen problem is the *Homology Groups of Binary 2D Image (HGB2I) Problem*: Given a binary 2D digital image, calculate the number of black connected components and the representative curves of the

[3] We refer to [20] for basic information in this area, to [23] for a comprehensive presentation and the web site [17] for the up-to-date information.

holes of these components. We can divide this problem in two sub-problems, H_0 problem (number of black connected components) and H_1 problem (number of holes). This problem and the way of computing and representing topological information (neighborhood, connectedness, orientation, etc.) form an important part in applications such as image classification, indexing, shape description and shape recognition.

This is not the first bio-inspired approach to problems to Algebraic Topology. In 1996, J. Chao and J. Nakayama connected Natural Computing and Algebraic Topology using Neural Networks [5] by extended Kohonen mapping. Some years after, Subramanian *et al.* presented in [3, 4] two works where Digital Image and Natural Computing were linked. Our paper can be seen as a new step from the work by Cristinal *et al.* [6, 7] in their effort for bridging Membrane Computing and Algebraic Topology.

The solution presented in this paper to the HGB2I problem has been designed in a new P system framework called *tissue-like P systems with catalysts*. It takes the membrane structure and symport-antiport communication rules with the power of catalysts and inhibitors.

Time to calculate the homology groups of 2D digital images with these P systems is logarithmic with respect to the input data with size n^2. This involves an improvement with regard to the algorithms development by S. Peltier et al. in [24], where they use irregular graphs pyramids with a time complexity of $\mathrm{O}(\mathrm{n}^{5/3})$.

The paper is organized as follows: In the next section we formally present the framework of *tissue-like P systems with catalysts*. In Section 3, we show how these P systems can be used to solve the H_0 and H_1 problems in Homology Theory. Next we will show a pair of examples. The paper ends with some final remarks and open lines for the future.

2 Tissue-like P Systems with Catalysts

Tissue P systems were presented in [15, 16]. This P system model is inspired in the intercellular communication and cooperation between neurons. The mathematical model of these devices is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules[4]. Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions.

In tissue-like P systems the membrane structure is a general undirected graph. The edges of such graph are not given explicitly, but they are deduced from the set of rules. From the seminal definition of tissue P systems, several research lines have been developed and other variants have arisen (see, for example, [1, 2, 10, 12, 14, 18, 22]).

[4] This way of communication for P systems was introduced in [21].

Catalytic P systems were introduced in [19]. The main feature of these P systems is the presence of objects in membranes such that they are not consumed by the application of the rule, but their presence in the membrane is necessary for the triggering. Catalysts have been deeply studied in Membrane Computing (see, e.g. [8, 13, 11]), but to the best of our knowledge, this is the first time in which catalysts are used for tissue P systems[5].

Next we provide the definition of tissue-like P systems with catalysts:

Definition 1. *A tissue-like P system with catalyst of degree $q \geq 1$ is a tuple of the form*

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

where:

1. *Γ is a finite alphabet, whose symbols will be called objects.*
2. *$\mathcal{E} \subseteq \Gamma$ is a finite alphabet representing the set of the objects in the environment available in an arbitrary large amount of copies.*
3. *$w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells in the initial configuration.*
4. *$\mathcal{R}$ is a finite set of enzymatic rules of the following form: $(\neg in, cat \,|\, i, u/v, j)$ for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, in, cat, $u, v \in \Gamma^*$. The length of a communication rule is defined as $|u| + |v|$. The catalyst and the inhibitor are not modified by the application of the rules and cat, in and v can be empty.*
5. *$i_0 \in \{0, 1, 2, \ldots, q\}$ denotes the output region, which can be the environment ($i_0 = 0$) or the region inside a cell ($1 \leq i_0 \leq q$).*

Informally, a tissue-like P system with catalysts of degree $q \geq 1$ can be seen as a set of q cells (each one consisting of a single membrane) labeled by $1, 2, \ldots, q$. The cells are the nodes of a virtual graph, where the edges connecting the cells are determined by the communication rules of the system (an edge linking two cells indicates that they are able to trade objects between them). In our definition, all objects in the alphabet can act as catalyst or inhibitor, depending on the applied rule. This means that the inhibitor or catalysts for a rule can be sent to another cell (or to the environment) by another rule.

The enzymatic rule $(\neg in, cat \,|\, i, u/v, j)$ can be applied over two cells (or a cell and the environment) i and j such that u (contained in cell i) is traded against v (contained in cell j). The rule is applied if in membrane with label i the objects of the set *cat* are present (catalyst) and none of the objects from the set *in* are present (inhibitors). If the catalyst and the inhibitor are empty, then the rule is called a *communication rule*.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered). In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e. in each step we apply a maximal multiset of rules.

[5] Comprehensive information about catalytic P systems can be found at [9].

A *configuration* is an instantaneous description of the system Π, and it is represented as a tuple $(w_0, w_1, \ldots, w_q)$. Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A sequence of computation steps is called a *computation*. A configuration is *halting* when no rules can be applied to it. The output of a computation is collected from its halting configuration by reading the objects contained in the output cell.

3 Using Tissue-like P Systems to Obtain H_0 and H_1

As pointed out above, given a binary 2D digital image, the problem consists on calculating the number of black connected components and the representative curves of the holes of these components. We can divide this problem in two sub-problems, the H_0 *problem* consists on calculating the number of black connected components and the H_1 *problem*, which consists on calculating the number of holes. In this paper we present a new technique to use tissue-like P systems with catalyst to obtain homological information of binary 2D digital images.

A 2D digital image I can be considered like a matrix where one pixel is an element of the matrix. If we have pixels the following question is to know when two pixels are *adjacent* (connected). There exists two natural possibilities, consider to work with a 4-adjacency (Von Neumann neighborhood in cellular automata) or 8-adjacency (Moore neighborhood in cellular automata).

In the first case, given a pixel K_{ij} (where $K = B \vee K = W$), the list of adjacent pixels to this is $\{K_{ij-1}, K_{ij+1}, K_{i-1j}, K_{i+1j}\}$ i.e.; the adjacent pixels to any pixel $K_{i,j}$ are just north, south, west, east of this (no in the diagonal respect to considered pixel), such we can observe in the following:

$$\begin{array}{ccc} & B & \\ B & K & W \\ & W & \end{array}$$

In the second we consider the pixel K_{ij} (where $K = B \vee K = W$), the list of adjacent pixels to this is $\{K_{i-1j-1}, K_{i-1j}, K_{i-1j+1}, K_{ij-1}, K_{ij+1}, K_{i+1j-1}, K_{i+1j}, K_{i+1j+1}\}$ i.e.; the adjacent pixels to a any pixel $K_{i,j}$ are just up, down, right and left of this and, moreover, we consider the diagonal objects, such we can observe in the following:

$$\begin{array}{ccc} B & W & B \\ W & K & B \\ B & B & W \end{array}$$

We have decide to consider in this paper the 4-adjacency for black pixels and the 8-adjacency for white pixels.

Given an image I with size n^2, we divide I in a set of pixels, black or white, but not both. We codify each pixel (i, j) by the object a_{ij} where $a = b$ (black) or $a = w$ (white). We can assign to each object a label associated to the codified pixel by this object. So, we have the objects of the form $(a_{ij}, (i,j))$. We will see below how to use these labels to solve our problem with P systems.

3.1 Solving H_0 Problem

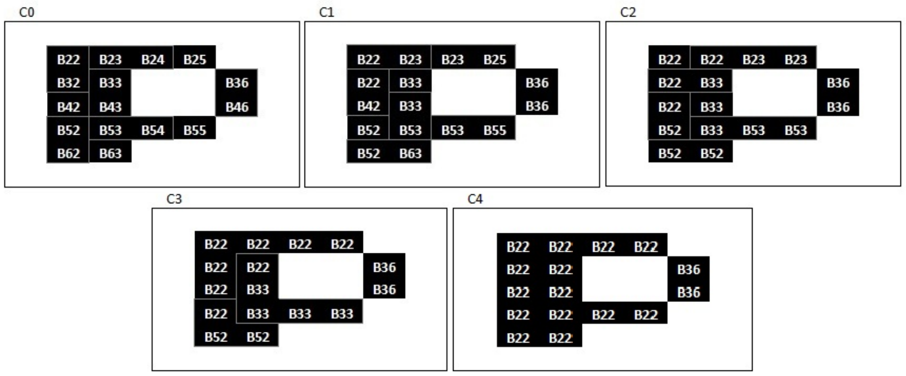


Fig. 1. A simple example to obtain H_0

In order to provide a logarithmic-time uniform solution to the H_0 problem, we design a family of tissue-like P systems with catalyst, Π_0. Given an image I of size n^2, we take the system of the family $\Pi_0(n)$ to work with I. The input data (image I) is codified by a set of objects b_{ij} and w_{ij} for $1 \le i, j \le n$. Each pixel of the image is given by an object z_{ij} with $z = b$ or $z = w$.

The family of P systems is defined as follows:

$$\Pi_0(n) = (\Gamma, \Sigma, \mathcal{E}, \omega_1, \omega_2, \mathcal{R}, i_{in}, i_0)$$

where:

- $\Gamma = \{a_i : 1 \le i \le n+2\} \cup$
  $\{b_{ij}, w_{ij} : 1 \le i, j \le n\} \cup$
  $\{(b_{ij}, (k,l)) : (1,1) \le (i,j) \le (k,l) \le (n,n)\} \cup$
  $\{A_{ijkl} : (1,1) \le (i,j) < (k,l) \le (n,n)\}$.
- $\Sigma = \{b_{ij}, w_{ij} : 1 \le i, j \le n\}$.
- $\omega_1 = \{a_1\}$.
- $\omega_2 = \emptyset$.
- $\mathcal{E} = \Gamma - \Sigma$.
- $\mathcal{R}$ is the set of rules:
  - $R_1 \equiv (1, a_i/a_{i+1}, 0)$ for $1 \le i \le n+1$.
    These rules generate a counter that will be used in the output of the system.

- ◦ $R_2 \equiv (1, b_{ij}/(b_{ij}, (i,j)), 0)$ for $1 \leq i, j \leq n$.
  These rules add labels to black pixels in order to work with them.
- ◦ $R_3 \equiv (1, (b_{ij}, (k,l))(b_{i'j'}, (k',l'))/(b_{ij}, (k,l))(b_{i'j'}, (k,l))A_{klk'l'}, 0)$ for $(1,1) \leq (k,l) < (k',l') \leq (n,n)$, and (i,j), (i',j') adjacent pixels.
- ◦ $R_4 \equiv (1, (b_{ij}, (k,l))(b_{i'j'}, (k',l'))/(b_{ij}, (k',l'))(b_{i'j'}, (k',l'))A_{k'l'kl}, 0)$ for $(1,1) \leq (k',l') < (k,l) \leq (n,n)$ and (i,j), (i',j') adjacent pixels.
  The two last types of rules change the labels of adjacent pixels, we need all the adjacent black pixels to have the same label, so we will know that they are all in the same connected component.
- ◦ $R_5 \equiv (A_{ijkl}|1, (b_{i'j'}, (k,l))/(b_{i'j'}, (i,j)), 0)$ for $1 \leq i, j, k, l, i', j' \leq n$.
  In these rules we introduce catalysts, and process becomes faster. The catalyst has been created when the pixel labeled by (k,l) traded its label for (i,j), so (i,j) and (k,l) are adjacent pixels and other pixels with these labels can be changed.
- ◦ $R_6 \equiv (a_{n+2}|1, (b_{ij}, (i,j))/\lambda, 2)$.
  With these rules we send one pixel for each connected component to the cell 2.

- $i_{in} = 1$ is the input cell.
- $i_0 = 2$ is the output cell.

Each system of the family implements the following stages:

1. *Label Allocation Stage*: Cell 1 trades objects b_{ij} against others with the form $(b_{ij}, (i,j))$ with the environment. The white objects are not transformed.
2. *Label Conversion Stage*: We can compare the black adjacent pixels by using catalyst, and we trade the label of the greatest pixel against the label of the other pixel; i.e. $(i, (b_{ij}, (i',j'))(b_{kl}, (k',l'))/(b_{ij}, (i',j'))(b_{kl}, (i',j'))A_{i'j'k'l'}, j)$, where (i,j) and (k,l) are adjacent pixels. Moreover, we can see a new object arriving to cell i. It is a catalyst and it is used to codify if two labels must be compared. Later, they are connected, and one of them can be changed by the other one, as we can see in the Figure 1.
3. *Answer Stage*: In the step $n+2$, the object a_{n+2} arrives to the cell 1 due to the counter. It is used by the system as a catalyst, and the objects with the form $(b_{ij}, (i,j))$ are sent to the output cell representing each one to a black connected component. The P system have used $n+2$ steps to obtain the number of black connected components of an n^2 image.

Figure 1 shows a computation of the $\Pi_0(7)$ system whose input data is the configuration $C0$ of the picture.

3.2 Solving H_1 Problem

With respect to the H_1 problem we use the same technique that we present above where labels are associated to the objects codifying pixels. We can construct a

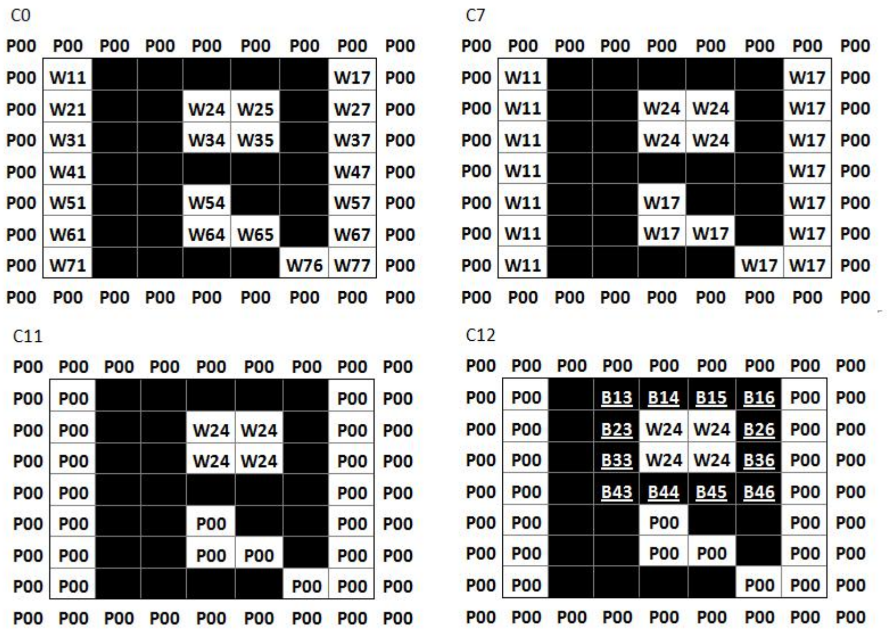


Fig. 2. Representative configurations of a simple example to obtain H_1

family of tissue-like P systems with catalyst, Π_1, to obtain a solution of the H_1 problem. Moreover, we can obtain the curves formed by black pixels containing the holes of the black connected components of the input image. So, we introduce in this paper a technique for segmenting images using catalysts.

Given an image I of size n^2 we take the system of the family $\Pi_1(n)$ to work with I. The input data (image I) is codified by a set of following objects: b_{ij} and w_{ij} for $1 \leq i, j \leq n$. Then, each pixel of the image is given by an object z_{ij} with $z = b \vee w$. The family of P systems is defined as follows:

$$\Pi_1(n) = (\Gamma, \Sigma, \mathcal{E}, \omega_1, \omega_2, \mathcal{R}_1, \dots, \mathcal{R}_{10}, \{\mathcal{R}_6, \mathcal{R}_8\} > \mathcal{R}_1, i_{in}, i_0)$$

where:

- $\Gamma = \{z_i : 1 \leq i \leq n+3\} \cup$
  $\{b_{ij}, \bar{b}_{ij}, w_{ij}, (w_{ij}, (k,l)) : 1 \leq i, j, k, l \leq n\} \cup$
  $\{(p_{ij}, (0,0)), (p_{ji}, (0,0)) : i = 0, n+1, 0 \leq j \leq n+1\} \cup$
  $\{Z_{ijkl} : (1,1) \leq (i,j) < (k,l) \leq (n,n)\}$.
- $\Sigma = \{b_{ij}, w_{ij} : 1 \leq i, j \leq n\}$.
- $\mathcal{E} = \Gamma - \Sigma$.
- $\omega_1 = \{z_1, (p_{ij}, (0,0)), (p_{ji}, (0,0)) : i = 0, n+1, 0 \leq j \leq n+1\}$.
- $\omega_2 = \emptyset$.

- The sets of rules are:
  - $R_1 \equiv (1, z_i/z_{i+1}, 0)$ for $1 \leq i \leq n+5$.
    This rule counts the number of steps of the process. We will use this to start the *Deleting Stage* after $n+2$ steps, and the *Segmenting Stage* after $n+4$ steps.
  - $R_2 \equiv (1, w_{ij}/(w_{ij}, (i,j)), 0)$ for $1 \leq i, j \leq n$.
    These are the only rules used in the *Label Allocation Stage*. These rules add labels to white pixels in order to work with them.
  - $R_3 \equiv (1, (w_{ij}, (k,l))(w_{i'j'}, (k',l'))/(w_{ij}, (k,l))(w_{i'j'}, (k,l))Z_{klk'l'}, 0)$ for $(1,1) \leq (k,l) < (k',l') \leq (n,n)$, $w_{ij}, w_{i'j'}$ adjacent pixels.
  - $R_4 \equiv (1, (w_{ij}, (k,l))(w_{i'j'}, (k',l'))/(w_{ij}, (k',l'))(w_{i'j'}, (k',l'))Z_{k'l'kl}, 0)$ for $(1,1) \leq (k',l') < (k,l) \leq (n,n)$, $b_{ij}, b_{i'j'}$ adjacent.
    These two set of rules are used in *Label Conversion Stage* to compare two adjacent white pixels, and change the label of one of them. We need all the adjacent white pixels to have the same label.
  - $R_5 \equiv (Z_{ijkl}|1, (w_{i'j'}, (k,l))/(w_{i'j'}, (i,j)), 0)$ for $1 \leq i, j, k, l, i', j' \leq n$.
    The catalyst Z_{ijkl} acts to become the process faster. It has been created when the pixel labeled by (k,l) traded its label for (i,j), so (i,j) and (k,l) are adjacent pixels and other pixels with these labels can be changed.
  - $R_6 \equiv (z_{n+3}|1, (p_{ij}, (0,0))(w_{kl}, (k',l'))/(p_{ij}, (0,0))(p_{kl}, (0,0))Z_{00kl}, 0)$ for $(i,j), (k,l)$ 8-adjacent pixels, $0 \leq i, j \leq n+1$, $1 \leq k, l, k', l' \leq n$.
    These rules are used in *Deleting Stage* to delete white pixels which are out of the connected black component. By using 8-adjacency, we become outer white pixels into pink pixels, in order to differentiate them from the interior white pixels (holes). We will refer to the objects p_{ij} as *pink* pixels.
  - $R_7 \equiv ((Z_{00ij}|1, (w_{i'j'}, (i,j))/(p_{i'j'}, (0,0)), 0)$.
    A new catalyst acts in the same way, trading white exterior pixel for pink pixels. In this way, the *Deleting Stage* takes only 2 step.
  - $R_8 \equiv (z_{n+5}|1, (w_{ij}, (i',j'))b_{kl}/(w_{ij}, (i',j'))\bar{b}_{kl}, 0)$ for w_{ij}, b_{kl} 8-adjacent pixels $1 \leq i'.j', i, j, k, l \leq n$.
    In the *Segmenting Stage* a black pixel is marked if it and a white pixel are 8-adjacent pixels. It starts after $n+2$ steps.
  - $R_9 \equiv (1, \bar{b}_{ij}/\lambda, 2)$ for $1 \leq i, j \leq n$.
    At the end, in the
  - *Answer Stage*, black marked pixels are sent to membrane number 2, so we obtain which black pixels are containing the holes.
  - $R_{10} \equiv (z_{n+6}|1, (w_{ij}, (i,j))/\lambda, 2)$ for $1 \leq i, j \leq n$.
    We want to obtain the number of holes too, so these rules send one white pixel for each hole to membrane number 2.
- $i_{in} = 1$ is the input cell.
- $i_0 = 2$ is the output cell.

We will also use priorities among rules. Rules from sets $\mathcal{R}_6$ and $\mathcal{R}_8$ are applied before rules from the set $\mathcal{R}_1$.

The computation of each P system of the family has the following phases:

1. *Label Allocation Stage*: Cell 1 trades objects w_{ij} against others with the form $(w_{ij}, (i, j))$ with the environment.
2. *Label Conversion Stage*: We compare the label of two white adjacent pixels, and we trade the label of the greatest pixel against the label of the other pixel; i.e., we use rules with the form $(i, (w_{ij}, (i', j'))(w_{kl}, (k', l'))/(w_{ij}, (i', j'))(w_{kl}, (i', j'))Z_{i'j'k'l'}, j)$, where (i, j) and (k, l) are adjacent pixels. Moreover, we can see a new object arriving to cell i, $Z_{i'j'k'l'}$. It is a catalyst and is used to codify when two labels must be compared. Then, the labels are connected, and one of them can be changed by the other one, as we can see in $C7$ in the Figure 2.
3. *Deleting Stage*: Initially, system keeps in cell 1 a set of objects codifying the frame of the input image ($p_{0i}, p_{n+1i}, p_{i0}, p_{in+1}$ for $i = 0, \ldots, n+1$) with the label $(0, 0)$ associated. When the input data is introduced in the system, the white pixels not contained inside of black connected components are sent to the environment to trade against of objects with the form of the frame. We need a linear number of steps with respect to n to eliminate all the possible white pixels. We can see the result in $C11$ in the Figure 2.
4. *Segmenting Stage*: This part begins when deleting stage finishes due to the counter z_i (rules R_1). If there are white pixels in cell 1 in this step are in a hole. The P system takes pairs of adjacent pixels, one black and the other white, adding a mark to the black pixels of these pairs. Then, we have marked the black pixels adjacent to a hole. We need a constant number of steps to segment an image with P systems. Figure 2 shows in $C12$ how the holes of the image are codified.
5. *Answer Stage*: We send the marked black pixels to output cell in the following step to be marked. So, we obtain, the representative curves of the holes in the image I. We also send white pixels which keep their labels, there is only one pixel for each connected white component, ie, for each hole in the image. We only need one step more with respect to the segmenting stage.

3.3 Complexity and Necessary Resources

Bearing in mind the size of the input data is $O(n^2)$, the amount of necessary resources for defining the systems of our two families and the complexity of our problems can be observed in the following table:

HGB2I Problem		
	$\mathbf{H_0}$ Problem	$\mathbf{H_1}$ Problem
Complexity		
Number of steps of a computation	$n+2$	$n+7$
Necessary Resources		
Size of the alphabet	$O(n^4)$	$O(n^4)$
Initial number of cells	2	2
Initial number of objects	1	$O(n)$
Number of rules	$O(n^6)$	$O(n^6)$
Upper bound for the length of rules of the systems	5	5

4 Final Remarks

Problems associated with the treatment of Digital Images have several interesting features from the Membrane Computing point of view. One of them is that they can be suitable for parallel processing. In many cases, the same sequential algorithm must be applied in different regions of the image which are independent. Other important feature is that the information of the image can be split into little pieces of information and the local transformations can be processed by re-writing-type rules.

These features lead us to explore the possibilities of using Membrane Computing techniques to well-known problems in Digital Images. In this paper we provide a solution in the framework of tissue P systems with catalysts, but a deeper study is necessary. The research lines related to the most suitable P system model for Homology Theory problems or which are the most relevant features of P systems which can represent the nature of the problems are open.

Acknowledgement

The first and second authors acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. The third and fourth authors acknowledge the support of the project MTM2006-03722 of the Ministerio español de Educación y Ciencia and the project PO6-TIC-02268 of Excellence of Junta de Andalucía.

References

1. Alhazov, A., Freund, R. and Oswald, M. Tissue P Systems with Antiport Rules ans Small Numbers of Symbols and Cells. *Lecture Notes in Computer Science*, **3572**, (2005), 100–111.

2. Bernardini, F. and Gheorghe, M. Cell Communication in Tissue P Systems and Cell Division in Population P Systems. *Soft Computing* **9**, 9, (2005), 640–649.
3. Ceterchi, R., Madhu, M., Paun, G., Subramanian, K.G. Array-rewriting P systems. *Natural Computing*, **2**, (2003), 229–249.
4. Chandra, P.H., Subramanian, K.G. On Picture Arrays Generated by P Systems, *Preproceedings of WMC6*, (2005), 282–288.
5. Chao. J. and Nakayama. J. Cubical Singular Simples Model for 3D Objects and Fast Computation of Homology Groups. *Proceedings of ICPR'96 IEEE*, (1996), 190–194 .
6. Christinal, H.A., Díaz-Pernil, D., Real, P. Segmentation in 2D and 3D Image Using Tissue-Like P System. *Lecture Notes in Computer Science*, **5856**, (2009) 169–176.
7. Christinal, H.A., Díaz-Pernil, D., Real, P. Using Membrane Computing for Obtaining Homology Groups of Binary 2D Digital Images. *Lecture Notes in Computer Science*, **5852**, (2009), 388–401.
8. Freund, R., Kari, L. Oswald, M., Sosik, P.Computationally universal P systems without priorities: two catalysts are sufficient, *Theoretical Computer Science*, **330**, (2005), 251–266.
9. Freund, R., Ibarra, O., Păun, A., Sosík, P., Yen, H.-C. Catalytic P Systems. In [23], 83–117.
10. Freund, R., Păun, Gh. and Pérez-Jiménez, M.J. Tissue P Systems with channel states. *Theoretical Computer Science*, **330**, (2005), 101–116.
11. Krishna, S.N. On Pure Catalytic P Systems. *Lecture Notes in Computer Science* **4135**, (2006), 152–165.
12. Krishna, S.N., Lakshmanan K. and Rama, R. Tissue P Systems with Contextual and Rewriting Rules. *Lecture Notes in Computer Science*, **2597**, (2003), 339–351.
13. Krishna, S.N., Păun, A. Results on catalytic and evolution-communication P systems. *New Generation Computing*, **22**, 4 (2004), 377–394.
14. Lakshmanan K. and Rama, R. On the Power of Tissue P Systems with Insertion and Deletion Rules. *Preproceedings of WMC*, (2003), 304–318.
15. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science*, **2387**, (2002), 290–299.
16. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. Tissue P systems. *Theoretical Computer Science*, **296**, (2003), 295–326.
17. P systems web page `http://ppage.psystems.eu/`
18. Prakash, V.J. On the Power of Tissue P Systems Working in the Maximal-One Mode. *Preproceedings of WMC*, (2003), 356–364.
19. Păun, Gh. Computing with Membranes. *Journal of Computer and System Sciences*. **61**, (2000), 108–143.
20. Păun, Gh. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
21. Păun, A. and Păun, Gh. The power of communication: P systems with symport/antiport. *New Generation Computing*, **20**, 3, (2002), 295–305.
22. Păun, Gh., Pérez-Jiménez, M.J. and Riscos-Núñez, A. Tissue P System with cell division. *BWMC2*, (2004), 380–386.
23. Păun, Gh. Rozenberg, G. Salomaa, A. eds: Handbook of Membrane Computing. *Oxford University Press*, (2009).
24. Peltier, S., Ion, A., Haxhimusa, Y., Kropatsch, W.G. and G. Damiand.: Computing Homology Group Generators of Images Using Irregular Graph Pyramids. *Lecture Notes in Computer Science*, **4538**, (2007), 283–294.

An Application of Genetic Algorithms to Membrane Computing

Gabi Escuela[1], Miguel A. Gutiérrez-Naranjo[2]

[1] Bio Systems Analysis Group
Friedrich Schiller University Jena
gabi.escuela@uni-jena.de
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
magutier@us.es

Summary. The process of designing a P system in order to perform a task is a hard job. The researcher has often only an approximate idea of the design, but finding the exact description of the rules is a heavy hand-made work. In this paper we introduce *PSystemEvolver*, an evolutionary algorithm based on generative encoding, that could help to design a P system to perform a specific task. We illustrate the use of *PSystemEvolver* with a simple mathematical problem: the computation of squared numbers.

1 Introduction

Natural Computing studies computational paradigms inspired from various well known natural phenomena in physics, chemistry and biology[3]. It abstracts the way in which nature computes, conceiving new computing models. The field is growing rapidly and there are many open research lines based on different aspects in which nature acts. Among them, *Cellular Automata* [16] conceived by Ulam and von Newman as a spatial distribution of cells able to reproduce the behavior of complex systems; *Genetic algorithms* introduced by J. Holland [13] which is inspired by natural evolution and selection in order to find a good solution in a large set of feasible candidate solutions; *Neural Networks* introduced by W.S. McCulloch and W. Pitts [15] it is based on the interconnections of neurons in the brain; *DNA-based* molecular computing, that was born when L. Adleman [2] published a solution to an instance of the Hamiltonian path problem by manipulating DNA strands in a lab; *Swarm Intelligence* [6] based on the behavior of mobile organisms as ants or bees communicating among them and acting in the environment; *Artificial Immune Systems* [5] based on the natural immune system

[3] An introduction on Natural Computing can be found in [12].

of biological organisms; *Amorphous computing* [1] inspired from the development of morphogenesis in biological organisms or *Membrane Computing* [17, 18] based on the functioning and morphology of living cells and tissues.

Membrane Computing was introduced by Gh. Păun in [17] under the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called *P systems*. Roughly speaking, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous nondeterministic maximally parallel manner.

The basic idea is to consider a distributed and parallel computing device structured in an arrangement of membranes which delimit compartments where various chemicals evolve according to local reaction rules. The objects can be eventually sent to the environment or to adjacent membranes under the control of specific rules. Because the chemicals from the compartments of a cell are swimming in an aqueous solution, the data structure we consider is that of a *multiset* – a set with multiplicities associated with its elements. Also, in close analogy with what happens in a cell, the reaction rules are applied in a parallel manner, with the objects to evolve by them and with the reactions themselves chosen in a non–deterministic manner.

In this way, we can define transitions from a configuration to another configuration of our system and hence we can define computations. A computation provides a result, for instance, in the form of the number of objects present in the halting configuration in a specified compartment, or in the form of a special object, `yes` or `no`, sent to the environment at the end of the computation (thus answering a decision problem that the system had to solve).

Evolutionary Algorithms (EAs) are generic population-based metaheuristics inspired in biological evolution to deal with combinatorial optimization problems. Four main EAs have been applied to different kind of problem domains: Genetic Algorithms, Genetic Programming, Evolutionary strategies and Evolutionary programming. Genetic Algorithms were introduced by J.H. Holland [13] an American psychologist and computer scientist who developed his theory to study self-adaptiveness in biological processes as well as to solve optimization problems. Concerning to functioning, a genetic algorithm is an iterative procedure which operates on a population where individuals are evaluated according a certain fitness value. Some individuals are selected according this value and produce offspring candidates which form the next generation[4]. For producing new individuals, two operators, namely crossover and mutation are used. Crossover takes two individuals called parents and produce one or two new individuals called offsprings. In its simplest form, it works by swapping pieces of information from the parents. The second operator is called mutation and it is applied by modifying an information unit in one individual according to a mutation rate.

In this paper we present a case study where genetic algorithms are used for designing a Membrane Computing device which performs a pre-fixed task. In the

[4] For details, see, for example [3].

literature, one can find several joint approaches of Membrane Computing and Genetic Algorithms as [14] or [4], but, to the best of our knowledge, this is the first time in which genetic algorithms are used to find a P system which solves an abstract computational problem.

The paper is organized as follows: Next we describe our case study for the application of Genetic Algorithms to the design of P systems. We start by describing an initial *population* of P systems and the genetic operators to act on them. In Section 3 we provide a short description of the genetic algorithm *PSystemEvolver* used in our experiments. In the following sections we provide the obtained experimental results. The paper ends with some final remarks and open lines for further research.

2 The Problem

The process of designing a P system in order to perform a task is a hard job. In many cases, the designer has an approximate idea of the membrane structure, initial multisets and set of rules necessary to describe the P system, but a little mistake in the description of the initial configuration or in the set of rules can leads to undesired consequences.

In this paper we present the case study of designing a P system which computes the square of a given number, e.g., number 4. To this aim, we will consider an initial *population* of P systems. Such population will *evolve* according to the natural selection of the evolution of alive beings (by means the corresponding *crossover* and *mutation* operations of a given genetic algorithm) and with the help of a *fitness* function we will obtain a member of the *population*, i.e., a P system obtained from the original ones, which performs the fixed task.

In order to perform our experiments, we consider that all the P systems have the same initial configuration. The allowed set of rules are of the types:

- **Evolution rules:** $[\,o \to u\,]_e$. The object o evolves to the multiset u in membrane with label e. Notice that u can be the *empty multiset* λ.
- **Dissolution rules:** $[\,r\,]_e \to s$. The object r dissolves the membrane e and goes to the surrounding region as object s. All the remaining objects in e also go to the surrounding region.

Our starting point is to consider a family of P systems $\mathbf{\Pi} = \{\Pi_i\}_{i \in I}$ where

$$\Pi = \langle \Gamma, H, \mu, w_e, w_s, R_i \rangle$$

- The alphabet $\Gamma = \{a, b, c, z_1, \dots, z_4\}$
- The set of labels $H = \{e, s\}$
- The membrane structure $\mu = [\,[\,]_e\,]_s$
- The initial multisets $w_e = a^2\, b\, z_1$ and $w_s = \emptyset$

The difference among the P systems in the family are the sets of rules R_i. In order to give a formal definition of $\mathbf{\Pi} = \{\Pi_i\}_{i\in I}$ we will start with a set of rules $\mathcal{R}$

$$\mathcal{R} = \left\{ \begin{array}{lll} r_1 \equiv [\,a \rightarrow\; a\,b\,]_e & r_7 \equiv [\,z_2 \rightarrow\; z_1\,]_e & r_{13} \equiv [\,a \rightarrow\; \lambda\,]_s \\ r_2 \equiv [\,b \rightarrow\; b\,c\,]_e & r_8 \equiv [\,z_3 \rightarrow\; z_4\,]_e & r_{14} \equiv [\,b \rightarrow\; \lambda\,]_s \\ r_3 \equiv [\,c \rightarrow\; b^2\,]_e & r_9 \equiv [\,z_1\,]_e \rightarrow b & r_{15} \equiv [\,b \rightarrow c\,]_e \\ r_4 \equiv [\,a \rightarrow\; b\,c\,]_e & r_{10} \equiv [\,z_2\,]_e \rightarrow a & r_{16} \equiv [\,c \rightarrow \lambda\,]_e \\ r_5 \equiv [\,z_1 \rightarrow\; z_2\,]_e & r_{11} \equiv [\,z_3\,]_e \rightarrow c & r_{17} \equiv [\,z_4 \rightarrow z_1\,]_e \\ r_6 \equiv [\,z_2 \rightarrow\; z_3\,]_e & r_{12} \equiv [\,z_4\,]_e \rightarrow a & r_{18} \equiv [\,z_4\,]_e \rightarrow b \end{array} \right\}$$

Our aim is to use genetic algorithms in order to find a P system which computes the square of number 4, from an initial set of P systems. The genetic evolution will only correspond to changes in the set of rules. The genetic operations in order to develop a genetic algorithm on the P systems are the following:

- **Crossover.** Given two P systems Π_1 and Π_2 and their sets of rules R_1 and R_2, let P_1^1 P_1^2 and P_2^1 P_2^2 two partitions of R_1 and R_2 respectively. Then, we obtain two offsprings Π_1' and Π_2' by considering the set of rules $R_1' = P_1^1 \cup P_2^1$ and $R_2' = P_1^2 \cup P_2^2$.
- **Mutation.** Given an evolution rule $[u \rightarrow v]_h$ with $u \in \Gamma$ and $v \in \Gamma^*$, the mutation operator changes the object u by one from $\Gamma - \{u\}$ or the object w in the multiset v by one object from $\Gamma - \{w\}$ or by λ. For an dissolution rule $[u]_h \rightarrow w$, the mutation operator changes the object u or w by a different one from Γ.

Only for practical reasons, in this case study we will impose an extra condition. All the P systems considered as individuals in our genetic algorithm must be deterministic. This is checked by ensuring that, for each P system and each membrane, there are no two rules triggered by the same object.

Example 1. Let us consider two P systems Π_1 and Π_2 and their sets of rules $R_1 = \{r_1^1, r_1^2\}$ and $R_2 = \{r_2^1, r_2^2, r_2^3\}$ with

$$\begin{array}{ll} r_1^1 \equiv [\,a \rightarrow\; a\,b\,]_e & r_2^1 \equiv [\,a \rightarrow\; b\,c\,]_e \\ r_1^2 \equiv [\,c \rightarrow\; b^2\,]_e & r_2^2 \equiv [\,z_4 \rightarrow z_1\,]_e \\ & r_2^3 \equiv [\,z_1\,]_e \rightarrow b \end{array}$$

Let P_1^1 and P_1^2 be a partition of R_1, $P_1^1 = \{r_1^1\}$ and $P_1^2 = \{r_1^2\}$ and P_2^1, P_2^2 a partition of R_2 with $P_2^1 = \{r_2^2, r_2^3\}$ and $P_2^2 = \{r_2^1\}$. Then, we obtain two offsprings Π_1' and Π_2' by considering the set of rules $R_1' = P_1^1 \cup P_2^1 = \{r_1^1, r_2^2, r_2^3\}$ and $R_2' = P_1^2 \cup P_2^2 = \{r_1^2, r_2^1\}$. Notice that, due to the restriction of determinism, we cannot get a new offspring by joining P_1^1 and P_2^2.

As example of the mutation operator, let us consider now the rule $[\,a \rightarrow\; b\,c\,]_e$. By applying a mutation rule we can obtain, for example, the rules $[\,z_1 \rightarrow\; b\,c\,]_e$ (changing a by z_1), the rule $[\,a \rightarrow\; b^2\,]_e$ (changing c by b) or $[\,a \rightarrow\; c\,]_e$ (changing b by λ).

Finally, we can describe the set of P systems $\mathbf{\Pi}$ considered as individuals for our genetic algorithm. A P system Π belongs to $\mathbf{\Pi}$ if it is a construct $\langle \Gamma, H, \mu, w_e, w_s, R_i \rangle$ as described above and the rules in R_i are from $\mathcal{R}$ or can be derived by a finite number of applications of the operators *crossover* and *mutation* from rules in $\mathcal{R}$.

3 The Genetic Algorithm

In this section we will briefly describe the genetic algorithm, *PSystemEvolver*, used in our case study. It follows the basic workflow:

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    Select the best individuals and produce new individuals (crossover and
    mutation operators)
    Evaluate the fitness of new individuals
    Generate the new population inserting the best individual
    from previous generations
end while
```

In order to apply the previous algorithm, we need to precise some details:

- The initial population consists on 30 individuals. In order to generate these individuals, 30 different random subsets of $\mathcal{R}$ are considered. The number of rules of each individual will not exceed 14, that is, the length of the alphabet times the number of membranes that we are considering. Before evaluating a possible solution using the fitness function, the P system is checked to assure determinism. If more than one rule in a specific membrane has the same right hand side (firing object), one of them is selected randomly to be active and the others are deleted from this P system.
- The fitness function is probably the key point in the application of the genetic algorithm for the design of P systems. In this case study we have considered a simple function: The absolute value of the difference between the number of objects c in the membrane s in the halting configuration of the P system and the expected number of such objects in an ideal found solution, i.e., 16 objects c. In order to prevent non-ending computations, we we limit to 20 the number of steps.
- The crossover and mutation operators will be applied on some randomized individuals with *good* score in the fitness function. For that, two parents are selected according to their fitness and mated to produce two offsprings that later could be mutated. Crossover and mutation rates of 0.8 and 0.8, respectively, have been considered in order to perform our experiments. We also varied this parameters to test the effect of this operators over the performance of the algorithm.

- As termination condition for the algorithm, a maximum of 30 generations has been considered.

4 Experimental Results

The chosen fitness function for our experiments with *PSystemEvolver* evaluates each P system according to its halting configuration. The computer simulations of all the computations have been performed by using the P-lingua simulator [9]. To calculate the fitness of each individual, *PSystemEvolver* generates the corresponding P-lingua file and call with it the simulator that produces a report file to obtain the evaluation for that P system.

To test the behavior of the algorithm, we performed 30 runs for each EA parameters setting. Table 1 shows the number of success for each experiment, that is, the number of times that *PSystemEvolver* could find a P system that solve the square(4) problem.

Experiment	**Crossover Rate**	**Mutation Rate**	**Successful Runs**
1	0.0	0.5	0/30
2	0.5	0.5	0/30
3	0.8	0.8	1/30
4	1.0	0.8	1/30

Table 1. Number of successful runs for different parameter settings.

Results demonstrated that is difficult to evolve a P system, even though initial configuration and membrane structure are fixed, and rules are provided for generating the initial population. This may be due to the fitness function that we considered for this problem, that conforms a landscape with a unique peak.

High values for crossover and mutation rates resulted beneficial for this algorithm, as can be seen in 1. Other types of mutations, as for example, rule activation or inactivation would be considered in future implementations.

The best P system P_{best} encountered by *PSystemEvolver* is described above by the rules R_{best}:

$$R_{best} = \left\{ \begin{array}{ll} r_1 \equiv [\, a \to a\, b\,]_e & r_5 \equiv [\, z_3 \to z_4\,]_e \\ r_2 \equiv [\, b \to b\, c\,]_e & r_6 \equiv [\, z_4\,]_e \to a \\ r_3 \equiv [\, z_1 \to z_2\,]_e & r_7 \equiv [\, a \to \lambda\,]_s \\ r_4 \equiv [\, z_2 \to z_3\,]_e & r_8 \equiv [\, b \to \lambda\,]_s \end{array} \right\}$$

5 Final Remarks

The advances in the research in Membrane Computing requires the design of more and more complex P systems. On the one hand, the theoretical research needs sophisticated designs which allows prove the ability of P systems for solving different type of problems by using a fixed ingredients (see, e.g., [10, 11]). On the other hand, Membrane Computing solutions to real-life problems needs to be quite precise in the design in order to find a sharp simulation of the processes [7, 8].

The design of such P systems is a hard task which must be performed by hand by the researcher. In this paper we explore the use of Genetic Algorithms as a help for designing P systems. The key point is finding a good fitness function. P systems are designed to make a computation and it is difficult to measure how far is the current design from the desired when the result of the computation is not the searched.

Many open questions arise from this work. As pointed above, the problem of finding the features of a *good* fitness function is open, but this is not the only one. A first research line involves a deeper study of the genetic algorithm operators, not only for the fitness function, but operators able also to modify the initial multisets or the membrane structures. A second line is related to the applications. In this paper we use a small theoretical problem, but the final target is to apply genetic algorithms for the design of complex P systems.

Acknowledgement

GE is supported by Universidad Simón Bolívar (Venezuela) and Deutscher Akademischer Austausch Dienst (DAAD) Grant A/08/94489.
MAGN acknowledges the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight Jr., R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM* **43**(5), (2000) 74-82.
2. L.M. Adleman. Molecular computations of solutions to combinatorial problems. *Science*, **226** (1994) 1021–1024.
3. M. Affenzeller, S. Winkler, S. Wagner, A. Beham. Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications. Chapman & Hall/CRC. (2009).
4. H. Cao, F.J. Romero-Campero, S. Heeb, M. Cámara, N. Krasnogor. Evolving cell models for systems and synthetic biology. *Systems and Synthetic Biology* **4**(1), (2010) 55–84.
5. L. de Castro and J. Timmis. Artificial Immune Systems: A New Computational Intelligence Approach. Springer, (2002).
6. A. Engelbrecht. Fundamentals of Computational Swarm Intelligence. Wiley and Sons, (2005).

7. G. Escuela, T. Hinze, P. Dittrich, S.Schuster, M. Moreno. Modelling Modified Atmosphere Packaging for Fruits and Vegetables using Membrane Systems. Accepted paper at Third International Conference on Bio-inspired Systems and Signal Processing BIOSIGNALS 2010. Valencia, Spain. January 2010.
8. T. Hinze, T. Lenser, G. Escuela, I. Heiland, S. Schuster. Modelling Signalling Networks with Incomplete Information about Protein Activation States: A P System Framework of the KaiABC Oscillator. *Lecture Notes in Computer Science* **5957**, (2010), 316-334.
9. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. An overview of P-Lingua 2.0. *Lecture Notes in Computer Science*, **5957** (2010), 264-288.
10. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Solving Subset Sum in linear time by using tissue P systems with cell division. *Lecture Notes in Computer Science* **4527** (2007), 170-179.
11. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* **83**(7), (2006) 593 - 611.
12. L. Kari and G. Rozenberg. The many facets of Natural Computing. *Communications of the ACM*, **51**(10), (2008) 72–83.
13. J.H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press. (1975)
14. L. Huang and N. Wang. An Optimization Algorithm Inspired by Membrane Computing. *Lecture Notes in Computer Science* **4222**, (2006) 49-52.
15. W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5** (1943) 115–133.
16. J. von Neumann. Theory of Self-Reproducing Automata. U. Illinois Press (1966).
17. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.
18. Gh. Păun. *Membrane Computing – An Introduction* Springer-Verlag, Berlin, 2002.

Applying Membrane Systems in Food Engineering

Gabi Escuela[1], Thomas Hinze[2], Peter Dittrich[2], Stefan Schuster[2], Mario Moreno-Álvarez[3]

[1] Dept. of Mathematics and Computer Science,
Bio Systems Analysis Group
Friedrich Schiller University Jena, Germany
gabi.escuela@uni-jena.de
[2] School of Biology and Pharmacy, Dept. of Bioinformatics
Friedrich Schiller University Jena, Germany
[3] Lab. de Biomoléculas, Universidad Simón Rodríguez
Canoabo, Venezuela

Summary. Food engineering deals with manufacturing, packaging and distributing systems for drug and food products. In this work, we discuss about the applicability of membrane systems to model environmental conditions and their effects on the produces during storage of fresh fruits and vegetables. In particular, we are interested in abstract molecular interactions that occur between produce, film and surrounding atmosphere factors involved in fresh fruit and vegetable package designs. We present a basic implementation to simulate the dynamical behaviour of these systems, due to gas exchanges and temperature fluctuations. Additionally, we reveal the benefits of this modelling approach and suggest some extensions as future directions to be considered.

1 Introduction

Membrane systems [17], also called P systems, had emerged to assist in the modelling of systems of concurrent reactions taking place in compartments, so as occur in biological systems. In this paper we use P systems as membrane structures delimiting compartments that contain multisets of objects representing molecules. The model was first presented in [3]. Compartments configuration changes over time (evolves) according to given rules that represent biochemical reactions and diffusions. In contrast to ODE-based approaches, each single molecule within the entire system is represented explicitly as individual entity. Capturing aspects of structural dynamics (changes in the membrane structure as well as in the composition of complex molecules) is seen as an advantageous feature of P systems. Inclusion of reaction kinetics into this formalism can be done by discretised kinetic laws [10]. We applied this mathematical formalism to a real known problem in fruits and vegetables post-harvest processing.

Fresh fruits and vegetables are living materials that continue to respire after harvesting exhibiting progressive biochemical changes. Food engineering methods to preserve freshness of post-harvest produces include low temperature storage and special packaging technologies, mainly Modified Atmosphere Packaging (MAP). MAP of fresh fruits and vegetables refers to the technique of enveloping the produce in a sealed container of polymeric film in order to modify the O_2 and CO_2 concentrations inside the package, reducing metabolic activity and increasing shelf life [16].

Designing MAP systems is a complex task that involves considerations about many interrelated environmental (as temperature and atmosphere composition), biological and package technology factors. Basic biological processes are respiration, transpiration, ethylene production and compositional changes due to metabolism. The variability of responses to internal and external signals depends on the characteristic of each plant organ type, developmental stage and physiological condition. In addition, much of the behaviour of a MAP system at cellular level are not fully understood. As examples we can refer to the little knowledge about the effect of CO_2 on the activity of respiratory enzymes [11]. Moreover, the contribution of the biochemical changes that alters physical properties of cell walls and tissues modifying the texture of the produce is not known in detail [9]. On the other hand, the mechanism of ethylene signal transduction that coordinates fruit ripening processes, is another aspect subject to study [1].

The difficulty to test different combination of gases and temperatures and the complexity of experimental setup for MAP systems had led to the development of various mathematical models [4, 11, 16, 19] and software [14]. In the literature, many respiration models are empirical fits of experimental data, based on one particular type and variety of fruit or vegetable, and most of them are based on the principles of enzyme kinetics and are represented using ODEs (for reviews see [5, 18]). However, there exists some lack on studies about the dynamical behaviour of these systems in terms of changes in environmental conditions, so as produce composition and physiology due to developmental processes [5]. It is worth mentioning that post-harvested fruits and vegetables, unlike other living materials, can be considered as less robust systems, as their responses on environmental fluctuations depends mostly on their actual configuration of biochemical components. In this context, some authors [7, 14] have considered the potential benefits of a systematic analysis or process-based modelling approach for fruits and vegetables. Considering that understanding the reaction network underlying MAP systems can give food experts more knowledge about emergent properties of packaged fruits and vegetables, we propose a framework based on membrane systems that abstracts basic biochemical reactions that occur in MAP systems. In the future, the proposed model can serve as a predictive tool to simulate changes in fresh produce on the molecular level, due to changes in environmental conditions.

This paper is organized as follows: firstly, biological and technical background of MAP systems is described in section 2. In Section 3 we present a P system framework for MAP, including the description of components, reaction kinetics and evolution of the system. Section 4 shows an application of the framework considering a package under modified atmosphere containing two produces. Finally, in Section 5 we point out some benefits of using our framework and future extensions of it.

2 Fresh Fruits and Vegetables Packaging

From the horticultural point of view, there are four main, sometimes overlapped, developmental stages identified in fruits, vegetables and flowers: growth, maturation, ripening and senescence. For packaging technology purposes, maturation and ripening stages of fresh commodities constitutes the central point of attention. During ripening, fruits and vegetables suffer considerable physiological changes that normally include: modification of colour through the alteration of chlorophyll, carotenoid, and/or flavonoid accumulation; textural modification via alteration of cell turgor and cell wall structure and/or metabolism; modifications of sugars, acids, and volatile profiles that affect nutritional quality, flavour, and aroma; and, generally enhanced susceptibility to opportunistic pathogens (likely associated with loss of cell wall integrity) [8]. The climacteric is a stage of fruit ripening associated with an autocatalytic production of ethylene, that rises cell respiration in some fruits. This physiological process marks the end of fruit ripening and the beginning of fruit senescence, in which a serie of irreversible events leads to breakdown and death of the plant cells.

In addition to temperature control, a reduced O_2 and elevated CO_2 atmosphere can extend the post-harvest life of whole and pre-cut commodities. This techniques reduce their respiration rate as well as production of ethylene, minimizing metabolic activity, delaying enzymatic browning and retaining visual appearance [13]. In order to obtain a good design of such a system, it is necessary to understand many concurrent reactions at a macro, meso and micro level, including the dynamics behind the interdependencies between environmental, biological and technical factors.

Basic environmental factors to be consider in MAP design are temperature, relative humidity (RH), and atmosphere composition. Other minor factors are light, chemicals as fungicides, growth regulators, ethylene perception blockers, etc. Temperature is the most important extrinsic factor affecting all elements of harvested produce. Considering packaged produces, temperature influences both the gas exchange of the produce and the permeability of the film for O_2, CO_2 and H_2O.

High RH in the atmosphere surrounding fruit diminishes dehydration and preserves freshness, whereas excessive RH may engender moisture condensation, microbial growth and decay of the produce. On the other hand, three gases in the surrounding atmosphere O_2, CO_2 and ethylene (C_2H_4), mostly influence stored fresh fruits and vegetables. Decreasing oxygen partial pressure can increase shelf life, but it is essential that the oxygen level not be reduced to the point that anaerobic respiration occurs. Anaerobiosis results in fermentation, the chemical conversion of carbohydrates into ethanol and organic acids, which may cause undesirable odours and flavours. Ethylene is a gaseous plant hormone (signal molecule) that regulates fruit ripening and senescence. O_2 is required for the synthesis of ethylene, while O_2 and low levels of CO_2 are required for its biological activity [9].

The most important biological processes occurring in fresh fruits and vegetables that affect them in packaging conditions over time are respiration, transpiration, ethylene production and compositional changes due to metabolism. Additionally, developmental processes, so as physiological and pathological breakdown should be considered.

Respiration involves a complex set of biochemical processes of which part is the oxidative breakdown of carbohydrates, lipids and organic acids into CO_2 and H_2O plus heat and metabolic energy. Respiration rate can be expressed in terms of O_2 consumed or CO_2 produced. The respiratory quotient (RQ), the ratio of CO_2 produced to O_2 consumed, ranges from about 0.7 to 1.4 depending on the substrate and its metabolic state (if the substrate is a lipid, $RQ < 1$, and $RQ > 1$ for organic acids)[5]. When carbohydrates are aerobically respired, the RQ is near 1, and the reaction is represented by Eq. (1).

$$C_6H_{12}O_6 + 6\,O_2 \rightarrow 6\,CO_2 + 6\,H_2O + energy \tag{1}$$

On the other hand, post-harvested fresh fruit and vegetables are mainly made up of water (80 to 95% approx.). Water loss is the primary cause of fresh weight loss and it is much more sensitive to changes in relative humidity around the commodity than to the rate of respiration[9]. Transpiration occurs due to the fact that fruits and vegetables internal atmosphere is saturated with water vapour, while external atmosphere contains lesser. Therefore, water loss rate depends on the external and internal water vapour pressure gradient.

There are two types of MAP techniques: passive or active. Passive MAP or equilibrium modified atmosphere consists in matching film permeation rates for O_2 and CO_2 with the respiration rate of the packaged produce. In some cases, it is likely that atmospheres within MAP will be actively established and adjusted, and this is realised creating a vacuum into the package and replacing the atmosphere with the desired gas mixture, and/or introducing gas absorbers/emmitters or other atmosphere-modifying elements into the package, so as using specialized films [18].

In a MAP design process, the type of material and its surface area and thickness are selected to obtain the desired equilibrium gas composition. Each film type has specific ranges of O_2 and CO_2 permeabilities, usually with the permeability of CO_2 being $3 - 5$ times that of O_2 [4]. Two strategies for creating film barriers exist: continuous and perforated films. Continuous film control movement of O_2 and CO_2 into or out of the package so that steady-state O_2 and CO_2 levels are achieved in the package, that is, they are used in the MAP design process assuming a constant respiratory rate of the produce. Perforated films with small holes are more suitable for produce having a high O_2 demand, and in this case the rate of gas exchange is a sum of gas permeation through the film and gas diffusion through the microperforations [9].

3 A P System-based MAP

We abstract a fruit or vegetable as a graph of cells or modules, like tissue P systems [15]. Each cell represents a compartment that contains species, and at a specific time, the contents of the compartment determines the cell configuration. This serves as a mechanism to differentiate one cell from other, given the possibility to create diverse tissue types, as occurs for example in fruits epicarp, mesocarp and endocarp tissues [7]. Additionally, as gas consumption-production occurs inside the cells, at the mitochondria level, and is stated that gas diffusion between cells depends on the geometry of the produce [11], differences in gas content in cells that conform a determinate region can adequately be represented. This is also in accordance to the idea that the ripening process usually starts in one region of a fruit and spreads to neighbouring regions, due to ethylene diffusion starting from promoter cells [1]. Produces into the package are represented as a population of membranes, giving the advantage that the model can deal with distinct fruits and vegetables within the same film, or the same produce in distinct developmental stages, varieties and/or presentations. Figure 1 shows as example, the schematic representation for such a system. In the next section we present the formal specification of our model.

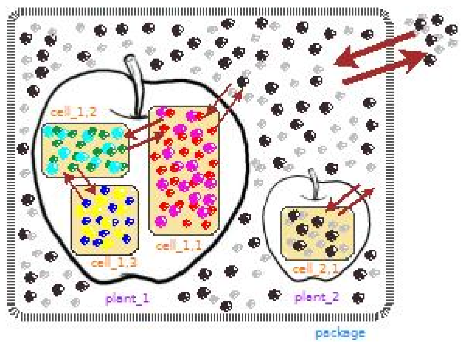


Fig. 1. A schematic representation for the MAP system model. In this case, two produces share a package: $plant_1$ is formed by three connected cells, and $plant_2$ is formed by a single cell. Arrows represent paths for molecules (spheres) diffusions

Multiset prerequisites

Let A be an arbitrary set and $\mathbb{N}$ the set of natural numbers including zero. A multiset over A is a mapping $F : A \longrightarrow \mathbb{N} \cup \{\infty\}$. $F(a)$, also denoted as $[a]_F$, specifies the multiplicity of $a \in A$ in F. Multisets can be written as an elementwise enumeration of the form $\{(a_1, F(a_1)), (a_2, F(a_2)), \ldots\}$ since $\forall (a, b_1), (a, b_2) \in F : b_1 = b_2$. The support $\text{supp}(F) \subseteq A$ of F is defined by $\text{supp}(F) = \{a \in A \mid F(a) > 0\}$. A multiset F over A is said to be empty iff $\forall a \in A : F(a) = 0$. The cardinality

$|F|$ of F over A is $|F| = \sum_{a \in A} F(a)$. Let F_1 and F_2 be multisets over A. F_1 is a subset of F_2, denoted as $F_1 \subseteq F_2$, iff $\forall a \in A : (F_1(a) \leq F_2(a))$. Multisets F_1 and F_2 are equal iff $F_1 \subseteq F_2 \wedge F_2 \subseteq F_1$. The intersection $F_1 \cap F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \min(F_1(a), F_2(a))\}$, the multiset sum $F_1 \uplus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = F_1(a) + F_2(a)\}$, and the multiset difference $F_1 \ominus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \max(F_1(a) - F_2(a), 0)\}$ form multiset operations. Multiplication of a multiset $F = \{(a, F(a)) \mid a \in A\}$ with a scalar c, denoted $\mathrm{c} \cdot F$, is defined by $\{(a, \mathrm{c} \cdot F(a)) \mid a \in A\}$.

P system components

Let $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ be the set of natural numbers without zero, and $m, n \in \mathbb{N}_+$. We define a P system for a MAP system as a construct:

$$\Pi_{\mathrm{MAP}} = (\mu, S, plant_1, \ldots, plant_m, G, L_0, D_1, \ldots, D_d, f_1, \ldots, f_d, \Delta\tau)$$

where:

- $\mu = [[[]_{cell_{1,1}} \cdots []_{cell_{1,n_1}}]_{plant_1} \cdots [[]_{cell_{m,1}} \cdots []_{cell_{m,n_m}}]_{plant_m}]_{package}$ is the spatial system structure composed of three inner levels: package, plants, and cells,
- S is a set of chemical species,
- $plant_1, \ldots, plant_m$ represent the produces into the package,
- G is a set of global parameters,
- $L_0 : S \rightarrow \mathbb{N}$ is a multiset of axioms representing the initial molecular configuration,
- D_ν is a diffusion (communication) rule among package and external environment ($\nu = 1, \ldots, d$),
- $f_\nu : (S \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ is a kinetic function attached to diffusion rule D_ν,
- $\Delta\tau \in \mathbb{R}_+$ is the time discretisation interval.

A diffusion rule D_ν can be of the form $[s] \rightarrow []s$ for molecules $s \in S$ leaving the package and released to the external environment, and $[]s \rightarrow [s]$ for molecules entering the package, respectively.

Furthermore, each $plant_i$ is defined as a tuple:

$$plant_i = (N_i, E_i, G_i, D_{i,1}, \ldots, D_{i,d_i}, f_{i,1}, \ldots, f_{i,d_i})$$

where:

- $N_i = \{cell_{i,1}, \ldots, cell_{i,n_i}\}$ defines a set of cells within plant i,
- $E_i \subseteq N_i \times N_i$ specifies a set of directed edges (diffusion channels between cells),
- G_i is a set of plant (organ) specific parameters,
- $D_{i,\kappa}$ represents a diffusion rule inside plant i and between plant i and package ($\kappa = 1, \ldots, d_i$),
- $f_{i,\kappa}$ is a kinetic function attached to diffusion rule $D_{i,\kappa}$.

Here, a diffusion rule can be of the form $[s]_{cell_{p,q}} \rightarrow []_{cell_{p,q}} s$ for molecules $s \in S$ leaving $cell_{p,q}$ and spread out into the package. A rule of the form $[]_{cell_{p,q}} s \rightarrow [s]_{cell_{p,q}}$ describes molecules entering $cell_{p,q}$ from the package. Finally, a rule of the form $[s]_{cell_{p,q}} \rightarrow [s]_{cell_{x,y}}$ formulates the directed transport of molecule s along the edge $(cell_{p,q}, cell_{x,y}) \in E_i$.

Each $cell_{i,j}$ is defined as a tuple

$$cell_{i,j} = (L_{i,j,0}, R_{i,j,1}, \ldots, R_{i,j,r_{i,j}}, f_{i,j,1}, \ldots, f_{i,j,r_{i,j}})$$

where:

- $L_{i,j,0} : S \rightarrow \mathbb{N}$ is a multiset of axioms representing its initial molecular configuration,
- $R_{i,j,k} = (A_{i,j,k}, B_{i,j,k})$ with $A_{i,j,k} : S \rightarrow \mathbb{N}$ (multiset of reactants) and $B_{i,j,k} : S \rightarrow \mathbb{N}$ (multiset of products) specifies a reaction rule including its stoichiometric factors,
- $\mathrm{f}_{i,j,k} : (S \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ is a function corresponding to kinetics of reaction $R_{i,j,k}$.

System evolution

A P system of the form Π_{MAP} evolves by successive progression of its configuration at discrete points in time $t \in \mathbb{N}$ for what we assume a global clock. Two consecutive dates t and $t+1$ specify a time span $\Delta\tau$. A system step at time t consists of three modification stages carried out from outer to inner spatial components of the system. Firstly, the diffusion between package and its environment is considered. To this end, the rules D_1 up to D_d are employed. Afterwards, the diffusion between package and cells as well as the intracellular diffusion is utilised by employing the rules $D_{i,\kappa}$ for each plant $i = 1, \ldots, m$. The last modification stage concerns application of the reaction rules specified in each cell. To cope with conflicts that can occur if the available amount of substrate cannot satisfy all matching diffusion and reaction rules, we prioritise all rules by their index: $D_1 > D_2 > \ldots > D_d$. Moreover, for each plant i: $D_{i,1} > D_{i,2} > \ldots > D_{i,d_i}$ and for each cell i, j: $R_{i,j,1} > R_{i,j,2} > \ldots > R_{i,j,r_{i,j}}$. Thus, we keep determinism of the system evolution and enable mass conservation. An alternative method for coping with conflicts is randomisation in selection and sequentialisation of diffusion and reaction rules.

The application of an arbitrary rule is organised into two consecutive steps. The first step identifies all molecules from the rule's left hand side acting as sources for diffusion or reactants. These molecules are removed from the current configurations. Corresponding molecules from the right hand side (destinations in case of diffusion and products in case of reactions) are then added.

We formulate discretised reaction-diffusion kinetics by specification of scalar functions $f : M \rightarrow \mathbb{N}$ based on a multiset $M : S \rightarrow \mathbb{N}$. Each function f converts the current configuration (L_t or $L_{i,j,t}$), a multiset of objects, into the number of turns for application of the corresponding diffusion or reaction rule. Here, kinetic laws $\hat{\mathrm{f}}(s)$ for each species $s \in S$ employ the multiplicity of its occurrences to formulate the corresponding reaction rate.

For updating the entire system configuration, we define an iteration scheme as shown in Figure 2. When this formalism is hidden in a software, the specification is intuitive and accessible for an expert focussing on MAP modelling.

4 Simulation

As a first application, we introduced as rules into the model only the basic processes involved in a MAP design: respiration and fermentation, so as gas diffusion between membranes. The influence of gas composition on respiration rates of produce has been widely represented by Michaelis Menten-type equation [5]. In this context, respiration rate is considered as a function of concentration in terms of enzymatic reaction, with O_2 in the place of substrate and the product CO_2 acting as inhibitor.

Temperature dependence over respiratory rate and over film permeability was represented using Arrhenius equation (Eq. 2).

$$k = F \times e^{-E_a/R \times T} \tag{2}$$

where E_a is the activation energy, expressed in joule per mol, defined as the energy that must be overcome for a chemical reaction to occur; R is the gas constant ($\approx 8.314\ J \cdot K^{-1} mol^{-1}$), T the absolute temperature, F is the pre-exponential factor that represents the total number of molecular collisions per second; and k corresponds to the number of collisions per second that result in a reaction. This can be related to the probabilistic approach to P systems introduced by [2] in order to obtain more biological-like models. In this context, the Arrhenius exponential term can be viewed as the probability per time unit that the reaction takes place.

In order to apply our model, we simulate the dynamical behaviour of an instance of a Π_{MAP} with two hypothetical fruits as it is shown in Fig. 1, using continuous film and passive MAP as package techniques. Rules that use symbol $\rightleftharpoons$ between reactants and products must been interpreted as reversible reactions. Into the formalism described in Fig. 2, a rule of the form $D_\alpha = [\sigma] \rightleftharpoons []\sigma$, for example, consists in the following two rules, in order of application: $[\sigma] \rightarrow []\sigma$ and $[]\sigma \rightarrow [\sigma]$.

Temperature is represented by T and expressed in Kelvin (K). O_2, CO_2 and H_2O abundances in the outside are represented by $O2ex$, $CO2ex$ and $H2Oex$ respectively. A and E symbolise the surface area in cm^2 and the thickness of the packaging film in mil ($1mil = 0.00254cm$). $pO2$ and $pCO2$ represent the reference film permeability in $mL \cdot mil \cdot cm^2 \cdot hr^{-1} \cdot atm^{-1}$ for O_2, CO_2 and H_2O, respectively. $EaO2$ and $EaCO2$ symbolise the permeability activation energy expressed in $J \cdot mol^{-1}$ for O_2 and CO_2, respectively. M_i symbolises mass of the produce i in kg. For simplicity, we assume that each cell in a produce has the same mass. $rO2$ and $rCO2f$ corresponds to the preexponential factor for produce respiration and fermentation in $mL \cdot kg^{-1} \cdot hr^{-1}$. $ErO2$ and $ErCO2f$ represent the respiration and fermentation activation energy for the produce expressed in $J \cdot mol^{-1}$.

Stage 1 (diffusion between package and external environment):

$\forall \alpha = 1, \ldots, d$

diffusion rule D_α	conditions	action
$[\sigma] \rightarrow []\sigma$	$(\sigma \in S) \wedge (\{(\sigma, f_\alpha)\} \subseteq L_t)$	$L_t := L_t \ominus \{(\sigma, f_\alpha)\}$
$[]\sigma \rightarrow [\sigma]$	$(\sigma \in S)$	$L_t := L_t \uplus \{(\sigma, f_\alpha)\}$

with $f_\alpha(L_t) = \left\lfloor k_\alpha(G) \cdot \varDelta\tau \cdot \hat{\mathrm{f}}(|L_t \cap \{(\sigma, \infty)\}|) \right\rfloor$

Stage 2 (diffusion between plant cells and package):

$\forall i = 1, \ldots, m$
  $\forall \alpha = 1, \ldots, d_i$

diffusion rule $D_{i,\alpha}$	conditions	action
$[\sigma]_{cell_{i,j}} \rightarrow []_{cell_{i,j}}\sigma$	$(\sigma \in S) \wedge (cell_{i,j} \in N_i) \wedge$ $(\{(\sigma, f_{i,\alpha})\} \subseteq L_{i,j,t})$	$L_{i,j,t} := L_{i,j,t} \ominus \{(\sigma, f_{i,\alpha})\}$ $L_t := L_t \uplus \{(\sigma, f_{i,\alpha})\}$
$[]_{cell_{i,j}}\sigma \rightarrow [\sigma]_{cell_{i,j}}$	$(\sigma \in S) \wedge (cell_{i,j} \in N_i) \wedge$ $(\{(\sigma, f_{i,\alpha})\} \subseteq L_t)$	$L_t := L_t \ominus \{(\sigma, f_{i,\alpha})\}$ $L_{i,j,t} := L_{i,j,t} \uplus \{(\sigma, f_{i,\alpha})\}$
$[\sigma]_{cell_{i,j}} \rightarrow [\sigma]_{cell_{i,k}}$	$(\sigma \in S) \wedge (k \neq j) \wedge (cell_{i,j} \in N_i) \wedge$ $(cell_{i,k} \in N_i) \wedge ((cell_{i,j}, cell_{i,k}) \in E_i)$ $\wedge(\{(\sigma, f_{i,\alpha})\} \subseteq L_{i,j,t})$	$L_{i,j,t} := L_{i,j,t} \ominus \{(\sigma, f_{i,\alpha})\}$ $L_{i,k,t} := L_{i,k,t} \uplus \{(\sigma, f_{i,\alpha})\}$

with $f_{i,\alpha}(L_t) = \left\lfloor k_{i,\alpha}(G, G_i) \cdot \varDelta\tau \cdot \hat{\mathrm{f}}(|L_t \cap \{(\sigma, \infty)\}|) \right\rfloor$

Stage 3 (reactions occurring within each cell):

$\forall i = 1, \ldots, m$
  $\forall j = 1, \ldots, n_i$
    $\forall \alpha = 1, \ldots, r_{i,j}$

reaction rule $R_{i,j,\alpha}$	conditions	action
$(A_{i,j,\alpha}, B_{i,j,\alpha})$	$f_{i,j,\alpha} \cdot A_{i,j,\alpha} \subseteq L_{i,j,t}$	$L_{i,j,t} := L_{i,j,t} \ominus f_{i,j,\alpha} \cdot A_{i,j,\alpha}$ $\uplus f_{i,j,\alpha} \cdot B_{i,j,\alpha}$

with $f_{i,j,\alpha}(L_{i,j,t}) = \left\lfloor k_{i,j,\alpha}(G, G_i) \cdot \varDelta\tau \prod_{\substack{\forall c \in \mathrm{supp}(A_{i,j,\alpha}) : \\ (R_{i,j,\alpha} = (A_{i,j,\alpha}, B_{i,j,\alpha}))}} \hat{\mathrm{f}}(|L_{i,j,t} \cap \{(c, \infty)\}|)^{|A_{i,j,\alpha} \cap \{(c,\infty)\}|} \right\rfloor$

Increment time t:

$L_{t+1} := L_t$
$\forall i = 1, \ldots, m$
  $\forall j = 1, \ldots, n_i$
    $L_{i,j,t+1} := L_{i,j,t}$

Fig. 2. Iteration scheme for the temporal evolution of $\varPi_{\mathrm{MAP}}$ system

Most of the values for these symbols were taken from the literature [4]. Symbols $O2$, $CO2$, $H2O$, $Ethanol$ and $Glucose$ represent amounts of species O_2, CO_2, H_2O, Ethanol and Glucose. Initial values for these symbols in each compartment and the rest of the parameters were assigned empirically.

$$
\begin{aligned}
S &= \{CO2, Ethanol, Glucose, H2O, O2\}\\
G &= \{M, T, R, A, E, CO2ex, H2Oex, O2ex, EaCO2, EaO2, pCO2, pO2, pH2O\}\\
N_1 &= \{cell_{1,1}, cell_{1,2}, cell_{1,3}\}\\
N_2 &= \{cell_{2,1}\}\\
G_i &: \{ErO2, rO2, ErCO2f, rCO2f\}\ \forall i \in \{1,2\}\\
D_1 &: []_{package}O2 \rightleftharpoons [O2]_{package}\\
\mathrm{f}_1(L_t) &= \left\lfloor \frac{A}{E} \cdot (\mathrm{O2ex} - \mathrm{L_t(O2)}) \cdot pO2 \cdot e^{\frac{-EaO2}{R \cdot T}} \right\rfloor\\
D_2 &: [CO2]_{package} \rightleftharpoons []_{package}CO2\\
\mathrm{f}_2(L_t) &= \left\lfloor \frac{A}{E} \cdot (\mathrm{L_t(CO2)} - \mathrm{CO2ex}) \cdot pCO2 \cdot e^{\frac{-EaCO2}{R \cdot T}} \right\rfloor\\
D_3 &: [H2O]_{package} \rightleftharpoons []_{package}H2O\\
\mathrm{f}_3(L_t) &= \left\lfloor \frac{A}{E} \cdot (\mathrm{L_t(H2O)} - \mathrm{H2Oex}) \cdot pH2O \right\rfloor\\
D_{1,1} &: []_{cell_{1,1}}O2 \rightleftharpoons [O2]_{cell_{1,1}} \qquad \mathrm{f}_{1,1}(L_t) = k_{1,1} \cdot L_t(\mathrm{O2})\\
D_{1,2} &: [CO2]_{cell_{1,1}} \rightleftharpoons []_{cell_{1,1}}CO2 \qquad \mathrm{f}_{1,2}(L_t) = k_{1,2} \cdot L_t(\mathrm{CO2})\\
D_{1,3} &: [H2O]_{cell_{1,1}} \rightleftharpoons []_{cell_{1,1}}H2O \qquad \mathrm{f}_{1,3}(L_t) = k_{1,3} \cdot L_t(\mathrm{H2O})\\
D_{1,4} &: [O2]_{cell_{1,1}} \rightleftharpoons [O2]_{cell_{1,2}} \qquad \mathrm{f}_{1,4}(L_t) = k_{1,4} \cdot L_t(\mathrm{O2})\\
D_{1,5} &: [CO2]_{cell_{1,2}} \rightleftharpoons [CO2]_{cell_{1,1}} \qquad \mathrm{f}_{1,5}(L_t) = k_{1,5} \cdot L_t(\mathrm{CO2})\\
D_{1,6} &: [H2O]_{cell_{1,2}} \rightleftharpoons [H2O]_{cell_{1,1}} \qquad \mathrm{f}_{1,6}(L_t) = k_{1,6} \cdot L_t(\mathrm{H2O})\\
D_{1,7} &: [O2]_{cell_{1,2}} \rightleftharpoons [O2]_{cell_{1,3}} \qquad \mathrm{f}_{1,7}(L_t) = k_{1,7} \cdot L_t(\mathrm{O2})\\
D_{1,8} &: [CO2]_{cell_{1,3}} \rightleftharpoons [CO2]_{cell_{1,2}} \qquad \mathrm{f}_{1,8}(L_t) = k_{1,8} \cdot L_t(\mathrm{CO2})\\
D_{1,9} &: [H2O]_{cell_{1,3}} \rightleftharpoons [H2O]_{cell_{1,2}} \qquad \mathrm{f}_{1,9}(L_t) = k_{1,9} \cdot L_t(\mathrm{H2O})\\
D_{2,1} &: []_{cell_{2,1}}O2 \rightleftharpoons [O2]_{cell_{2,1}} \qquad \mathrm{f}_{2,1}(L_t) = k_{2,1} \cdot L_t(\mathrm{O2})\\
D_{2,2} &: [CO2]_{cell_{2,1}} \rightleftharpoons []_{cell_{2,1}}CO2 \qquad \mathrm{f}_{2,2}(L_t) = k_{2,2} \cdot L_t(\mathrm{CO2})\\
D_{2,3} &: [H2O]_{cell_{2,1}} \rightleftharpoons []_{cell_{2,1}}H2O \qquad \mathrm{f}_{2,3}(L_t) = k_{2,3} \cdot L_t(\mathrm{H2O})\\
R_{i,j,1} &: Glucose + 6\ O2 \rightarrow 6\ CO2 + 6\ H2O\ \ \forall i \in \{1,2\} \wedge j \in \{1,2,3\}\\
\mathrm{f}_{i,j,1}(L_{i,j,t}) &= \left\lfloor \frac{L_{i,j,t}(Glucose)}{\Theta_{i,j,1,1} + L_{i,j,t}(Glucose)} \cdot \frac{L_{i,j,t}(\mathrm{O2})^6}{\Theta_{i,j,1,2} + L_{i,j,t}(\mathrm{O2})^6} \cdot \frac{M_i}{3} \cdot rO2 \cdot e^{\frac{-ErO2}{R \cdot T}} \right\rfloor\\
R_{i,j,2} &: Glucose \rightarrow 2\ Ethanol + 2\ CO2\ \ \forall i \in \{1,2\} \wedge j \in \{1,2,3\}\\
\mathrm{f}_{i,j,2}(L_{i,j,t}) &= \left\lfloor \frac{L_{i,j,t}(\mathrm{Glucose})}{\Theta_{i,j,2,1} + L_{i,j,t}(\mathrm{Glucose})} \cdot M_i \cdot rCO2f \cdot e^{\frac{-ErCO2f}{R \cdot T}} \right\rfloor
\end{aligned}
$$

Figure 3 shows the corresponding courses of $plant_1$ internal gas composition, resulting from following parameter setting for the discrete iteration scheme: $A = 100$, $E = 1$, $M_i = 0.1$, $pO2_i = 1620000$, $EaO2_i = 43100$, $pCO2_i = 238000$, $EaCO2_i = 34300$, $rO2_i = rCO2f_i = 3 \times 10^{14}$, $ErO2_i = ErCO2f = 70700$, $pH2O_i = 1$, $\Theta_{i,j,1,k} = 1$, for $i \in \{1, 2\}$ and $j \in \{1, 2, 3\}$ and $k \in \{1, 2\}$; $k_{1,j} = 0.2$ for $j \in \{1, \ldots, 9\}$, $k_{2,j} = 0.2$ for $j \in \{1, \ldots, 3\}$. A fixed value $T = 277.15$ was considered for a constant temperature scenario, and transient values for $273.15 \leq T \leq 293.15$ were obtained through a sigmoid function to represent changes in temperature over time in another scenario. Simulations have been performed using Copasi [12]. Differences in internal gas composition of $plant_1$ have been observed

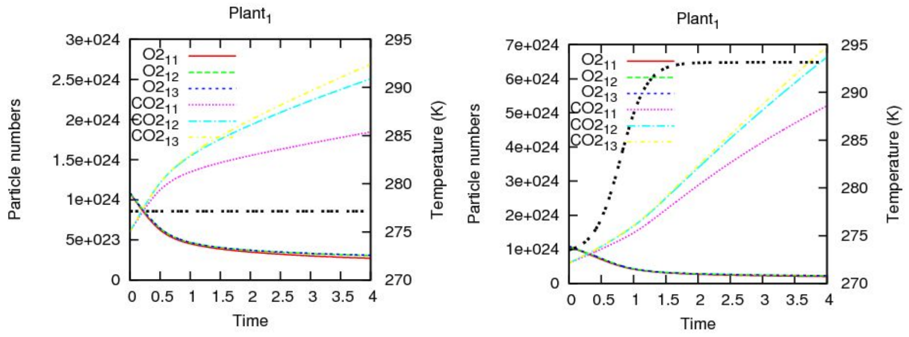


Fig. 3. Dynamical behaviour for gas composition for $plant_1$ in constant and varying temperature scenarios

during time due to the interplay between cellular respiration and fermentation processes and intercellular diffusion. Those differences could determine the form of maturation of the produce, in this case, from the center to the skin. An equilibrium is reached in the package gas composition, while respiration rates of the produces diminished.

5 Conclusions

Using a membrane based model for MAP, we presented a framework that is able to abstract packaging for different fruit and vegetable types, varieties or developmental stages. Respiration of the produce is considered as the basic process when modelling MAP, and predictions about the dynamical behaviour of such systems can be improved taking into account environmental, biological and technical factors. Our approach allows extensions including other low level processes, such as ethylene signaling pathway, cell/tissue rupture due to produce cutting and transport of other molecules, that can been advantageously modeled using P systems.

Finally, the quality of the packaged produce (taste, odour, texture, colour and appearance) is based on some subjective consumer evaluation. These traits are based on specific product properties, such as sugar content, volatile production and cell wall structure [19], and therefore can be introduced into the model through reactions, as a mechanism to obtain more knowledge about the impact of packaging conditions over product quality.

Considering texture (softening) of the fruit or vegetable, a next extension of this model will include cell wall structure contents and reactions. To do so, we plan to add a new membrane surrounding the structure on the cell level in order to represent this compartment, and a new stage in the evolution algorithm. Also, dissolution rules can be defined in this framework to capture cell wall dissolution. Additionally, to perform further simulations, we plan to implement the model in P-Lingua [6], basically because ODE-based simulators for membrane systems are unable to deal with structural changes.

Acknowledgements

GE is supported by Universidad Simón Bolívar (Caracas, Venezuela) and Deutscher Akademischer Austausch Dienst (DAAD) Grant A/08/94489.

References

1. Alexander, L. and Grierson, D. (2002). Ethylene biosynthesis and action in tomato: a model for climacteric fruit ripening. *J. Exp. Bot.*, 53(377):2039–2055.
2. Ardelean, I. and Cavaliere, M. (2003). Modelling biological processes by using a probabilistic P system software. *Natural Computing*, 2(2):173–197.
3. Escuela, G., Hinze, T., Dittrich, P., Schuster S., and Moreno, M. (2010). Modelling Modified Atmosphere Packaging for Fruits and Vegetables using Membrane Systems. In A. Fred, J. Filipe, H. Gamboa (Eds.), Proceedings of the Third International Conference on Bio-inspired Systems and Signal Processing (BIOSIGNALS2010). IEEE Engineering in Medicine and Biology Society, Institute for Systems and Technologies of Information Control and Communication, ISBN 978-989-674-018-4, pp. 306-311, INSTICC Press, 2010.
4. Exama, A., Arul, J., Lencki, R., Lee, L., and Toupin, C. (1993). Suitability of plastic films for modified atmosphere packaging of fruits and vegetables. *J Food Sci*, 58(6):1365–1370.
5. Fonseca, S., Oliveira, F., and Brecht, J. (2002). Modelling respiration rate of fresh fruits and vegetables for modified atmosphere packages: a review. *J Food Eng*, 52(2):99–119.
6. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., and Riscos-Núñez, A. An overview of P-Lingua 2.0. Lecture Notes in Computer Science, **5957** (2010), 264-288.
7. Génard, M., Bertin, N., Borel, C., Bussieres, P., Gautier, H., Habib, R., Lechaudel, M., Lecomte, A., Lescourret, F., Lobit, P., and Quilot, B. (2007). Towards a virtual fruit focusing on quality: modelling features and potential uses. *J Exp Bot*, 58(5):917–928.

8. Giovannoni, J. (2004) Genetic regulation of fruit development and ripening, The Plant Cell Online, 16(90001):170–180.
9. Gross, K., Wang, C., and Saltveit, M. (2004). The commercial storage of fruits, vegetables, and florist and nursery stocks. *Agriculture Handbook*, 66.
10. Hinze, T., Lenser, T., and Dittrich, P. (2006). A protein substructure based P system for description and analysis of cell signalling networks. *Lect Notes Comput Sci*, 4361:409–423.
11. Ho, Q., Verboven, P., Verlinden, B., Lammertyn, J., Vandewalle, S., and Nicolai, B. (2008). A continuum model for metabolic gas exchange in pear fruit. *PLoS Computational Biology*, 4(3):e1000023.
12. Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006). Copasi—a complex pathway simulator. *Bioinformatics*, 22(24):3067–3074.
13. Kader, A. and Zagory, D. and Kerbel, E. (1989). Modified atmosphere packaging of fruits and vegetables. Crit Rev Food Sci Nutr, 28(1):1–30.
14. Mahajan, P., Oliveira, F., J., M., and Frias, J. (2007). Development of user-friendly software for design of modified atmosphere packaging for fresh and fresh-cut produce. *Innovative Food Science and Emerging Technologies*, 8(1):84–92.
15. Martín-Vide, C., Păun, G., Pazos, J., and Rodríguez-Patón, A. (2003). Tissue P systems. *Theor Comput Sci*, 296(2):295–326.
16. Paul, D. and Clarke, R. (2002). Modelling of modified atmosphere packaging based on designs with a membrane and perforations. *J Membr Sci*, 208(1–2):269–283.
17. Păun, G. and Rozenberg, G. (2002). A guide to membrane computing. *Theor Comput Sci*, 287(73–100):73–100.
18. Rodriguez-Aguilera, R. and Oliveira, J. (2009). Review of design engineering methods and applications of active and modified atmosphere packaging systems. *Food Engineering Reviews*, 1(1):66–83.
19. Tijskens, L., Hertog, M., and Nicolaï, B. (2001). *Food Process Modelling*. Woodhead Pub.

How Does a P System Sound?

Manuel García-Quismondo, Miguel A. Gutiérrez-Naranjo,
Daniel Ramírez-Martínez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
mangarfer2@alum.us.es, magutier@us.es,
thebluebishop@gmail.com

University of Sevilla. Avda. Reina Mercedes s/n, 41012, Sevilla, Spain

Summary. P systems are computational devices versatile enough to represent many real-life scenarios. In this paper, we present a first interpretation for P systems where a computation produces a set of sounds. The idea is to associate sounds to the application of specific rules in the P system. The application of such rules produces sounds of one time unit. Different rules produce different sounds. The combination of such sounds along time can be interpreted as *music*.

1 Introduction

Membrane Computing is a nice symbolic game inspired by Nature. It abstracts the processes taking place in the compartmental structure of a living cell, interpreting them as computing operations. The information is usually represented as multisets of metabolites placed in compartments (membranes). According to different types of rules, the chemical compounds can evolve or be sent to other membranes. Depending on the model, the membranes can be created, divided or dissolved. These cellular devices are called P systems.

The description of a P system is usually made by a n-tuple which enumerates the different sets of components. Such sets usually involve alphabets for the symbols and labels, the multisets placed initially in the different regions and rules. Depending on the model, the description can also involve an alphabet for describing electrical charges or an explicit description of the membrane structure.

The aim of P systems is to perform changes on the multisets placed on the membranes (and, eventually, changes in the membrane structure) according to the set of rules. The rules are usually applied in a synchronous maximally parallel manner. It means that we can consider a clock which measures the time, taking the application of all the rules exactly one time unit. Rules are applied in parallel in a double way: rules are applied in all the membranes simultaneously and inside each membrane, all the objects that can evolve according to the rules must do it.

A configuration is an instantaneous description of a P system. After one time unit, several rules have been applied and the configuration has changed, so we obtain a new configuration. If no rules can be applied, we have a *halting configuration* and the computation stops.

In order to perform such changes, we need a *syntax* expressive enough to represent the different configurations of the P system, but we also need a *semantics* which allows us to apply the rules in an appropriate way. Changes in the syntax and/or the semantics produce the large panoply of P system models.

By considering the syntax and the semantics of a P system we can calculate the configurations which are reachable from a given configuration. Due to the inherent non-determinism of P systems, if one object can trigger more than one rule, one of them is chosen and the computation goes on. If we consider all the configurations to be reachable from a given one in each step, we obtain the so-called computation tree. In this abstract level, we can study many interesting problems related to confluence, reachability of configurations or halting criteria; problems inherent to all computation models.

Nonetheless, not only are we usually interested in the intrinsic properties of P systems themselves, but in using them as tools for dealing with other processes as well. Such processes can be of a symbolic nature, as solving the SAT [7] problem, or real-world problems such as the simulation of a population of bearded vultures [2] or the study of the epidermal growth factor receptor [8] in cells.

The design of P systems for dealing with these problems needs the explicit description of the syntax and the semantics of the problem, but we also need an *interpretation* of the symbols in our problem. According to the problem, the occurrence of a symbol in a membrane can be interpreted as a scavenger bird [1] in an ecosystem or a metabolite in the vesicle of a cell [4], but it also can represent a length unit of the width of the truck of a tree [9] or the negation of a literal in a propositional formula [7].

Other elements of the description of a P system can also different interpretations according to the problem which describes. In many situations, the electrical charges of the membranes act as *traffic lights*: a generated object can be several unit times *waiting* for a change in the polarization of a membrane which allows it to cross the membrane. Another typical example is to use a sequence of objects $\{z_1, \ldots, z_n\}$ as a *counter* till the appearance of the objects z_n which produces an event (maybe the dissolution or division of the membrane) exactly after n steps.

In this paper we provide a new interpretation to the application of a rule. We propose the use of membranes to produce sounds and, in a broad sense of the word, to produce music. The use of membranes to produce music should not be something surprising: from the origin of the Mankind, the vibration of tight membranes has been a source of sound. Nonetheless, to the best of our knowledge, this is the first time in which P systems are seen as *musical instruments*.

In the literature, there exist several approaches between music and computational biology, as [5] or [6] among others, but this is the first time in which P systems and music are brought together.

The paper is organized as follows: First we give some ideas about the interpretation of some computational events as sound producers. Next we provide some details about the implementation used in our experiments. In Section 4 we show an example of a P system designed for producing sounds and we finish with some final remarks.

2 Generating sounds

The word *sound* has two main meanings. On the one hand, it can be defined as the vibration transmitted through an elastic media (solid, liquid or gas), with frequencies in the approximate range of 20 to 20,000 hertz. The second meaning is related to the sensation stimulated in the organs of hearing by such vibrations in the air or other medium. Figure 1 shows the notes that can be usually reproduced by human voice or instruments.

	Oc. 0	Oc. 1	Oc. 2	Oc. 3	Oc. 4	Oc. 5	Oc. 6	Oc. 7	Oc. 8
C		32,70	65,41	130,81	261,63	523,25	1046,50	2093,00	4186,01
C#		34,65	69,30	138,59	277,18	554,37	1108,73	2217,46	
D		36,71	73,42	146,83	293,66	587,33	1174,66	2349,32	
D#		38,89	77,78	155,56	311,13	622,25	1244,51	2489,02	
E		41,20	82,41	164,81	329,63	659,26	1318,51	2637,02	
F		43,65	87,31	174,61	349,23	698,46	1396,91	2793,83	
F#		46,25	92,50	185,00	369,99	739,99	1479,98	2959,96	
G		49,00	98,00	196,00	392,00	783,99	1567,98	3135,96	
G#		51,91	103,83	207,65	415,30	830,61	1661,22	3322,44	
A	27,50	55,00	110,00	220,00	440,00	880,00	1760,00	3520,00	
A#	29,14	58,27	116,54	233,08	466,16	932,33	1864,66	3729,31	
B	30,87	61,74	123,47	246,94	493,88	987,77	1975,53	3951,07	

Fig. 1. Frequency in hertzs of the notes

In this paper we explore the relationship between P systems and music. In a broad sense, playing music consists on the production of vibration of a media (sounds) during several time units. The appropriate combination of the frequencies of the vibration (different notes) along time produces (eventually) a harmonious sound.

From this point of view, producing sounds with P systems is based on three key ideas:

- As we have seen above, time is generally considered to be discrete in P systems. We usually consider an universal clock for all the events (evolution of objects, creation, division or dissolution of membranes ...). Such a universal clock can be considered as a uniform pulse. In other words, if we want to see a P system as a sound generator, we already have a metronome.

- We can associate a *sound* to an event produced in a P system in a time unit. The association is arbitrary and can be seen as an *interpretation*. In the same way as we can associate a *meaning* to a membrane, an object or to an electrical charge and see them as a geographical region, as a bearded vulture or as the signal of a traffic light, we can also associate a sound to an event produced during the computation.
  The main difference is that we associate a sound to the *application* of the rule, not to the rule itself. In such a way, the application of the rule is an event which is produced in one step of computation *at time t*. It makes sense to associate a sound to the application of a rule. The duration of the sound will be also arbitrarily prefixed. We will consider that the application of a rule produces one sound during a *time unit*.
- The third key point is the parallelism inherent in most of the P system models. As pointed above, in a time unit, several rules can be applied in different membranes. If we associate a sound to the application of one rule, we will produce several sounds *simultaneously*. In other words, for a musical piece, the application of some rules can produce the main voice and simultaneously, an accompanying *chord* can be performed. The parallelism of P systems opens a door to explore the possibilities of P systems as tools for producing polyphony.

3 Implementation

A first implementation of these ideas has been carried out by using the P system simulator SCPS [3]. This simulator was written in Prolog and one of its features is that the user can obtain a file which includes a report with the applied rules in each time unit.

3.1 Log parsing

These files can be easily parsed by using regular expressions or any other text search method, so a new *output format* can be defined in order to make the sound generation easier. Just to test this, three different musical interpretations can done by processing the *events*.

- **First method:** A different frequency is assigned to each membrane in the system. In this method, we consider that the membrane *vibrates* at the associated frequency when an object cross it, regardless of the object.
- **Second method:** A different frequency is assigned to each element. In this method, the vibration of one membrane does not depend on the membrane itself, but on the object which crosses the membrane.
- **Third method:** A different frequency is assigned to each rule. In this case, the frequency is associated to a membrane and an object, but it can be considered in a more general case, since other rules different from communication ones (dividing, dissolving, evolving, . . .) can also have an associated frequency.

3.2 Frequency and timing assignment

Time length for each musical event from a P system is an arbitrary choice as long as the human ear can hear each separate note. If the length of an event is too short, the output wave will be a valid physical recreation of the sonic wave, in terms of frequencies and amplitudes, but from an human point of view, it will be just a burst of beeps with no musical sense at all. Typical assignments for the first generated waves have been in the order of 0.1 seconds to 1 second. Greater lengths are, for now, just boring.

The frequency assignment for the events is, in a cold way, an arbitrary task too. Anyway, as far as somebody has to hear and to give an interpretation to the sounds, it seems to be quite obvious that a *friendly* assignment must be done. These assignments can be:

- **Harmonic frequencies:** Starting from an arbitrary frequency, the next ones are selected so they are integer multiples of that first one. So, if 440 Hz. is selected (central A), the next frequencies must be 880, 1320, 1760, ... and so on. Integer dividers are allowed too.
- **Occidental dodecaphonic notes:** As it has been described above, each musical note in the Western culture is defined by a well-known frequency. If each, or some, of these frequencies are assigned to different events, any known chromatic melody can be played by the P system.
- **Occidental musical scales:** Well defined and largely used in the musical world, musical scales are subsets of the dodecaphonic system. In a scale, only some notes are picked from the entire the set, following any rule (typically, a set of intervals from a starting note). This way the scales of *C minor*, *B flat major*, *F diminished* or any other one, can be defined.

Depending on what target to be reached, different assignments can be done.

- For *Musical P systems*, those which have been specifically designed to be interpreted as music event generators, all chromatic range or any scale will be a good choice, depending on the target melody.
- For *Non Musical P systems*, it seems to be a good idea to avoid reinforcing any possible inherent cacophony in the interpretation. As far as it is completely unknown how a P system sounds (until now), there is nothing such as a *correct assignment*, so harmonic frequencies or any scale notes are assigned. This way, at least, a *tuned* melody will be generated regardless of its musical coherence.

3.3 Wave generation

The process to generate a wave which represents how a P system sounds involves these steps:

1. Parse report (log) file from SCPS.
2. Select an interpretation (scales, harmonics,...).
3. Select parameters (time lengths, overlapping behavior, intensity levels,...).

4. Render the wave according to interpretation and parameters.
5. Post-processing the wave so as to make it *easy listened.*

The parsing is done by using regular expressions, which are a widely known tool for text search. The interpretation and the parameters are defined for harmonic and chromatic sounds inside the scripts as variable values. The render is accomplished following these steps:

1. Total length is calculated.
2. For each simulation step, every applied rule generates a single tone wave.
3. These single event waves are summed and the resulting wave can be normalized in amplitude. This exploits the parallelism, so chords are the interpretation of the set of rules which are fired in a step.
4. After every summed waves are linked together, the final wave can be normalized.
5. The resulting wave is saved as a PCM file with a sampling frequency of 44.100 Hz. and 16 bits, ready to be played in any standard WAV or music player.

4 Experimental Results

The first experiment performed was to consider a P system designed for an specific task without any relationship to music. The chosen one was a family of P system which solves the Subset Sum problem. We chose an instance of the problem, provided the corresponding input to the P system and got the information of the applied rules by using the simulator.

The obtained report file was processed by associating arbitrary sounds to the application of rules. The performance of this piece was presented during the Eight Brainstorming Week on Membrane Computing, held in Seville in the first days of February of 2010. The obtained sound was absolutely cacophonous, but it was the first answer to the question that entitles this paper.

The next challenge was to design a P system whose unique target was to produce sounds in a harmonious way. This first design is showed below. It was called Π_{HB}. From a technical point of view, it is a very simple P system. It only uses two types of rules:

- *Chemical decomposition or analysis:* $[\,a \to b\,c\,]_e$ The object a inside the membrane with label e is decomposed into two new objects b and c.
- *Osmotic reaction:* $[\,a\,]_e \to a\,[\,]_e$ The object a is sent out from the partially-permeable membrane with label e.

This first design Π_{HB} does not exploit the parallelism of P systems. In fact it is a deterministic P system which works sequentially, but illustrates the possibilities of P systems for producing sounds. In a more complex design, other membranes or objects could also perform parallel calculus and hence, producing the corresponding chords.

Let us consider now the following P system $\Pi_{HB} = (\Gamma, H, \mu, E, w_1, w_2, R)$ where

- $\Gamma = \{g_3, a_3, b_3, c_4, d_4, e_4, f_4, g_4, A_1, A_2, B_1, \ldots, B_4, C_1, \ldots, C_{11}, D_1, D_2, E_1, \ldots, E_4, F_1, F_2, G_1, \ldots, G_6, T_1, z_1, \ldots, z_{26}\}$ is the working alphabet.
- $H = \{1, 2\}$ is the set of labels.
- $\mu = [\,[\,]_2\,]_1$ is the membrane structure.
- $w_1 = \emptyset$ and $w_2 = \{g_3\, G_1\}$ are the initial multisets.
- R is the set of rules:

$$\begin{array}{llll} [g_3]_2 \to g_3\,[\,]_2 & [a_3]_2 \to a_3\,[\,]_2 & [b_3]_2 \to b_3\,[\,]_2 & [c_4]_2 \to c_4\,[\,]_2 \\ [d_4]_2 \to d_4\,[\,]_2 & [e_4]_2 \to e_4\,[\,]_2 & [f_4]_2 \to f_4\,[\,]_2 & [g_4]_2 \to g_4\,[\,]_2 \end{array}$$

$$\begin{array}{llll} [A_i \to a_3\, A_{i+1}]_2 & i \in \{1,3,5\} & [B_i \to b_3\, B_{i+1}]_2 & i \in \{1,2,3,5\} \\ [A_2 \to a_3\, G_2]_2 & & [B_4 \to b_3\, G_4]_2 & \\ [A_4 \to a_3\, G_6]_2 & & [B_6 \to b_3\, A_5]_2 & \\ [A_6 \to a_3\, F_1]_2 & & & \end{array}$$

$$\begin{array}{llll} [C_i \to c_4\, C_{i+1}]_2 & i \in \{1,3,4,5,7, & & \\ & \quad 9,11,12,13\} & & \\ [C_2 \to c_4\, B_1]_2 & & [D_i \to d_4\, D_{i+1}]_2 & i \in \{1,3\} \\ [C_6 \to c_4\, G_8]_2 & & [D_2 \to d_4\, C_3]_2 & \\ [C_8 \to c_4\, B_5]_2 & & [D_4 \to d_4\, C_{11}]_2 & \\ [C_{10} \to c_4\, D_3]_2 & & & \\ [C_{14} \to c_4]_2 & & & \end{array}$$

$$\begin{array}{llll} [E_i \to e_4\, E_{i+1}]_2 & i \in \{1,3\} & [F_1 \to f_4\, F_2]_2 & \\ [E_2 \to e_4\, C_7]_2 & & [F_2 \to f_4\, E_3]_2 & \\ [E_4 \to e_4\, C_9]_2 & & & \end{array}$$

$$\begin{array}{ll} [G_1 \to g_3\, A_1]_2 & \\ [G_i \to g_3\, G_{i+1}]_2 & i \in \{2,4,6,8,9\} \\ [G_3 \to g_3\, C_1]_2 & \\ [G_5 \to g_3\, A_3]_2 & \\ [G_7 \to g_3\, D_1]_2 & \\ [G_{10} \to g_4\, G_{11}]_2 & \\ [G_{11} \to g_4\, E_1]_2 & \end{array}$$

4.1 Overview of the computation

The initial configuration consists on two membranes: An outer membrane with label 2 and an inner membrane with label 1. At the beginning, the membrane with label 1 is empty ($w_1 = \emptyset$) and the membrane with label 2 only has two objects: g_3 and G_1 ($w_2 = \{g_3\, G_1\}$), so the initial configuration can be expressed as $C_0 \equiv [\,[\,g_3\, G_1\,]_2\,]_1$.

From this initial configuration two rules can be applied: $[g_3]_2 \rightarrow g_3\,[\,]_2$ and $[G_1 \rightarrow g_3\,A_1]_2$. Both are applied simultaneously in one time unit. The object g_3 is sent out from membrane 2 and goes to membrane 1 and object G_1 is decomposed into objects g_3 and A_1. The new configuration at time $t = 1$ is $C_1 \equiv [\,[\,g_3\,A_1\,]_2\,g_3\,]_1$. Object g_3 in membrane 1 does not evolve any more, since there is no rule for it in the membrane 1. Nonetheless, objects in membrane 2 evolve according to the rules $[g_3]_2 \rightarrow g_3\,[\,]_2$ and $[A_1 \rightarrow a_3\,A_2]_2$. We obtain $C_2 \equiv [\,[\,a_3\,A_2\,]_2\,g_3^2\,]_1$, i.e., two copies of the object g_3 are placed in membrane 1 and objects a_3 and A_2 are placed in membrane 2. Bearing in mind that objects in membrane 2 do not trigger any rule, the remaining configurations can be obtained by applying simultaneously a chemical rule and an osmotic rule. Membrane 2 in configuration 46 contains objects c_4 and C_{14}. Rules $[c_4]_2 \rightarrow c_4\,[\,]_2$ and $[C_{14} \rightarrow c_4]_2$ are then applied and membrane 2 has only object c_4 in the configuration 47. Again $[c_4]_2 \rightarrow c_4\,[\,]_2$ is applied and finally, in configuration 48, membrane 2 is empty. No more rules can be applied, and the computation stops.

4.2 Interpretation

In order to obtain a melody, we associate a sound to an *event*. In this case, we choose to associate a note to the application of one *osmotic rule*. In this way, if the rule $[a]_2 \rightarrow a\,[\,]_2$ is applied at time t, then one sound will be produced during one time unit starting at time t. The remaining rules do not produce sounds. In order to simplify the interpretation, the objects sent out by osmotic rules have a natural interpretation in music. Such objects are g_3, a_3, b_3, c_4, d_4, e_4, f_4 and g_4. They are naturally interpreted by considering c_4 as the central note C in a keyboard piano. According to the computation, the first objects which cross out membrane e by application of the corresponding osmotic rules are g_3, g_3, a_3, etc. Figure 2 shows the melody obtained by the computation where the considered time unit corresponds to one quaver[1]

5 Final Remarks

In this paper we have presented a first approach between P systems and music. The idea of associating a sound to an event is quite natural and opens a large panoply of possibilities. The first one is to explore the possibilities of polyphony, since the parallelism is inherent to Membrane Computing devices. Other possibilities can be to introduce different timbres (different instruments) or different lengths of notes. Another exciting way to explore is the non determinism of P systems, since it can produce different melodies depending on the chosen computation.

[1] The G-clef and the time signature have been added. They do not correspond to the interpretation of the P system computation.

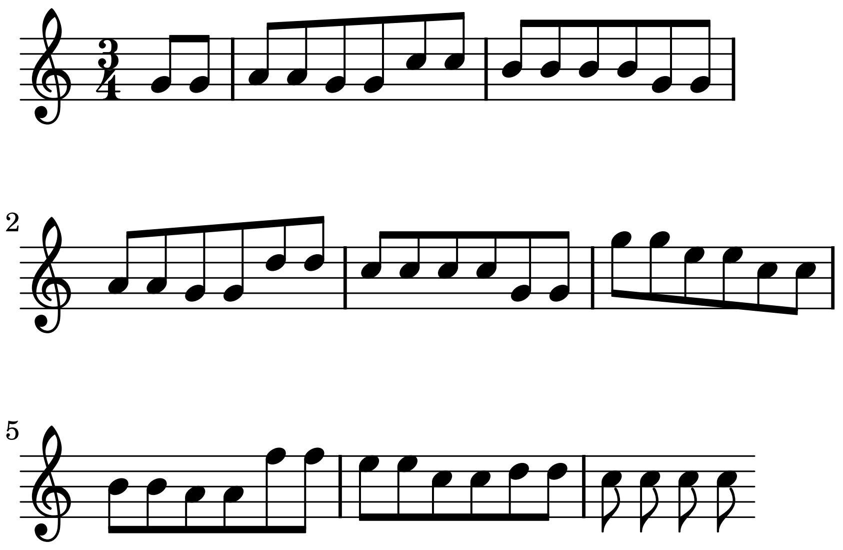


Fig. 2. The interpretation of the computation of the P system Π_{HB}

Acknowledgement

The second author acknowledges the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A P system based model of an ecosystem of some scavenger birds. *Lecture Notes in Computer Science*, **5957** (2010), 182-195
2. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, D. Sanuy, A. Margalida. Modeling ecosystem using P systems: The bearded vulture, a case study. *Lecture Notes in Computer Science*, **5391** (2009), 137-156.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. A Simulator for Confluent P Systems. Third Brainstorming Week on Membrane Computing. RGCN Report 01/2005 Fénix Editora, Sevilla (Spain), (2005), 169-184 .
4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero. How to express tumours using membrane systems. *Progress in Natural Science*, **17**(4) (2007), 449-457.

5. R. King, C. Angus. PM - Protein Music. *Computer Applications in the Biosciences*, **12**, (1996), 251-252.
6. C. Miner F. Della Villa. DNA Music. *The Science Teacher*, **64**(5), (1997), 19–21.
7. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division. Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, Computer and Automaton Research Institute of the Hungarian Academy of Sciences (2003), pp.:284-294.
8. M.J. Pérez-Jiménez, F.J. Romero-Campero. A study of the robustness of the EGFR signalling cascade using continuous membrane systems. *Lecture Notes in Computer Science*, **3561** (2005), 268-278.
9. A. Romero-Jiménez, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez. Graphical Modelling of Higher Plants Using P Systems. *Lecture Notes in Computer Science* **4361**, (2006) 496-506.

Membrane Computing Meets Artificial Intelligence: A Case Study

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

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

Summary. The usual way to find a solution for a NP complete problem with Membrane Computing techniques is by brute force algorithms where all the feasible solutions are generated and they are checked simultaneously by using massive parallelism. These solutions work from a theoretical point of view but they are implementable only for small instances of the problem. In this paper we provide a family of P systems which brings techniques from Artificial Intelligence into Membrane Computing and apply them to solve the N-queens problem.

1 Introduction

Brute force algorithms have been widely used in the design of solutions for NP problems in Membrane Computing. Trading time against space allows us to solve NP problems in polynomial (even lineal) time with respect to the input data. The cost is the number of resources, mainly the number of membranes, which grows exponentially. The usual idea of such brute force algorithms is to encode each feasible solution in one membrane. The number of candidates to solution is exponential in the input size, but the coding process can be done in polynomial time. Once generated all these candidates, each of them is tested in order to check whether it represents a solution to the problem or not. This checking stage is made simultaneously in all membranes by using the massive parallelism inherent to Membrane Computing. Any computational device that performs this checking sequentially needs an exponential amount of time. After the checking stage ends, the P system halts and sends a signal to the user with the output of the process.

Such theoretical process works and many different P system models have been explored by searching the limits between tractability and intractability [2]. In such way, several semantic and syntactic ingredients have been mixed and nowadays there exist many open questions and open problems in the area (see, e.g., [6]).

In spite of the great success in the design of theoretical solutions to NP problems, these solution have an intrinsic drawback from a practical point of view. It is not clear yet whether Membrane Computing will have an *in vitro*, *in vivo* or *in silico* implementation, but in any case, a membrane will have a *space* associated (maybe a piece of memory in a computer, a pipe in a lab or the volume of a bacteria) and brute force algorithms only will be able to implement little instances of such problems. As an illustration, if we consider an in vivo implementation where each feasible solution is encoded in an elementary membrane and such elementary membrane is *implemented* in a bacteria of mass similar to E. Coli ($\sim 7 \times 10^{-16}$ kg., see [8]), then, a brute force algorithm[1] which solve an instance of a NP problem with input size 40 will need approximately the mass of the Earth for an implementation ($\sim 6 \times 10^{24}$ kg., *ibid.*).

In this paper we explore the possibility of searching solutions to NP problems with Membrane Computing techniques, but taking ideas from Artificial Intelligence instead of using brute force algorithms. Of course, the worst case of any solution of an NP-problem needs and exponential amount of resources, but we are not always in the worst case. The contribution of using search strategies from Artificial Intelligence is that, on average, the number of resources for solving several instances of an NP problem decreases with respect to the number of resources used by brute force, since an exponential number of resources is always used in the former one. The case study is the N-queens problem (Section 2), which was previously studied in the framework of Membrane Computing in [3].

The paper is organized as follows: Next we present the N-queens problem and recall the brute force algorithm presented in [3]. In Section 3, we give a brief notions of searching strategies in Artificial Intelligence and in Section 4, we present an implementation of depth-first search with P systems. In Section 5, we present a family of P systems which solve the N-queens problem based on the cellular implementation. Finally, some conclusions and new open research lines are presented.

2 The N-queens Problem

Along this paper we will consider the N-queens problem as a case study. The N-queens problem is very popular among computer scientists. It is a generalization of a classic problem known as the 8-queens problem. The original one is attributed to the chess player Max Bezzel and it consists on putting eight queens on an 8×8 chessboard in such way that none of them is able to capture any other using the standard movement of the queens in chess, i.e., at most one queen can be placed on each row, column and diagonal line.

The 8-queens problem was later generalized to the N-queens problem, with the same rules but placing N queens on a N×N board.

[1] A similar comparison was proposed by Niall Murphy during the Tenth Workshop on Membrane Computing.

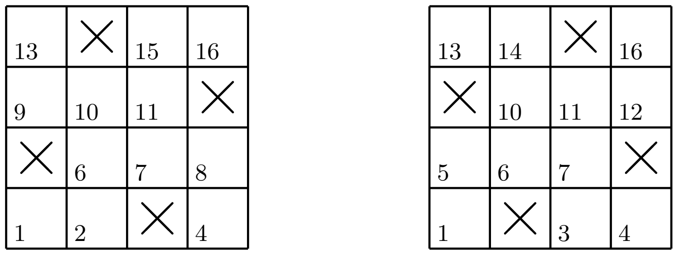


Fig. 1. Solutions to the 4-queens problem

In [3], a first solution to the N-queens problem in Membrane Computing was shown. For that aim, a family of deterministic P systems with active membranes was presented. In such family, the N-th element of the family solves the N-queens problem and the last configuration encodes *all* the solutions of the problem.

In order to solve the N-queens problem, a truth assignment such that it makes true a formula in CNF is searched. This problem is exactly SAT, so the solution presented in [3] uses a modified solution for SAT from [5].

In such a paper, it was proven that given an integer $N \geq 3$, there exists a formula Φ in conjunctive normal form such that encodes the N-queens problem with N^2 variables and $\frac{1}{3}(5N^3 - 6N^2 + 4N)$ clauses.

Some experiments were presented by running the corresponding P systems with an updated version of the the P-lingua simulator [1]. The experiments were performed on a one-processor Intel core2 Quad (with 4 cores at 2,83Ghz), 8GB of RAM and using a C++ simulator over the operating system Ubuntu Server 8.04.

In the 3-queens problem, three queens should be placed on a 3×3 chessboard. According to our representation, the problem can be expressed by a formula in CNF with 9 variables and 31 clauses. The *input multiset* has 65 elements and the P system has 3185 rules. Along the computation, $2^9 = 512$ elementary membranes need to be considered in parallel. Since the simulation was carried out in a one-processor computer, in the simulation, these membranes were evaluated sequentially. It took 7 seconds to reach the halting configuration. It is the 117-th configuration and in this configuration one object `No` appears in the environment. As expected, this means that we cannot place three queens on a 3×3 chessboard satisfying the restriction of problem.

In the 4-queens problem, we try to place four queens on a 4×4 chessboard. According to our representation, the problem can be expressed by a formula in CNF with 16 variables and 80 clauses. Along the computation, $2^{16} = 65536$ elementary membranes were considered in the same configuration and the P system has 13622 rules.

The simulation takes 20583 seconds (> 5 hours) to reach the halting configuration. It is the 256-th configuration and in this configuration one object `Yes` appears in the environment. This means that there exists at least one solution to the problem. In order to know such solutions, we check the multiset of the elementary membranes. In this case there are two elementary membranes in the halting configuration with the following multisets:

$$w_1 = \{f_1, f_2, t_3, f_4, t_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, t_{12}, f_{13}, t_{14}, f_{15}, f_{16}\}$$

$$w_2 = \{f_1, t_2, f_3, f_4, f_5, f_6, f_7, t_8, t_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, t_{15}, f_{16}\}$$

Such multisets encode the solution showed in the Figure 1

According to this design, for the solution of the N-queens problem in a standard 8×8 chessboard 2^{64} = 18.446.744.073.709.551.616 elementary membranes should be considered simultaneously. If we follow with the analogy from the Introduction, an *E. Coli implementation* of such P system we will need approximately a metric tone of bacteria to solve the problem. Does it means that Membrane Computing is not able to find at least one solution to the N-queens problem on a 8×8 chessboard?

3 Searching Strategies

Searching has been deeply studied in Artificial Intelligence. In its basic form, a *state* is an instantaneous description of the world and two states are linked by a *transition* which allows us to reach a state from a previous one. In such way, we consider a directed graph where the nodes are the states and the edges are the actions. Giving a starting state, a sequence of actions (a path in a graph) to one of the final states is searched.

In sequential algorithms, only one node is considered in each time unit and the order in which we explore new nodes determines the different searching strategies. In the usual framework, several possible unexplored nodes are reachable and we need to choose one of them in order to continue the search. In the best case, we have a heuristic which can help us to decide the best options among the candidates. Such heuristic represents, in a certain sense, how far the considered node is from a solution node and it captures our information about the nature of the problem. In many other situations we have no information about how far we are from a solution and we need to use a *blind strategy.*

Since there is no information about the nature of the problem, blind strategies are based exclusively in the topology of the graph and the order in which new nodes are reached.

There exists a clear parallelism between the space of states represented as a directed graph and the computation trees in Membrane Computing. In Membrane Computing, we start with an initial description of the world (the initial configuration) and, in the general case, we have several sets of applicable rules which lead us to different configurations. We choose one of the reachable configurations and go on with the process till reaching a halting configuration. In the case of recognizer P systems, no matter which new configuration we choose among the different possibilities since all of them lead us to the same answer, but this is not the general case.

The two basic blind search strategies are depth-first search and breath-first search. The main difference between them is that depth-first search follows a path to its completion before trying an alternative path. Some path can be infinite, so this search may never succeed. It involves *backtracing*: One alternative is selected for each node and it backtracks to the next alternative when it has pursued all of the paths from the first choice. In the worst case, depth-first search will explore all of the $O(b^m)$ nodes in the search tree, where m is the maximum depth of any node and b is the maximum branching factor. The complexity in time is $O(b^m)$, and the complexity in space is $O(bm)$.

In breath-first search the order in which nodes are explored depends on the number of arcs in the path. The algorithm always selects one of the paths with fewest arcs. If there exists a solution at depth d, the total number of generated nodes is $O(b^{d+1})$. In this case the complexity in time and in space is $O(b^{d+1})$.

4 Depth-first Search with P Systems

The idea of representing an instantaneous description of the world as a state and a step from a state to the following one as an edge in the graph is so general that many real-life problems can be modeled as a problem of space of states. In this paper we present a first approach to depth-first search with P systems. The aim is to show that Membrane Computing provides all the ingredients that we need to find a solution for any problem represented as a space of states and hence, to be a useful tool to solve many real-life problems.

The aim of this first approach is not minimalist. We are not looking for the minimum number of ingredients for implementing in P systems the depth-first search. In fact, we use four of the most powerful available ingredients: inhibitors, cooperation, priorities and dissolution. As we will remark in Section 6, it is an open question to weaken these conditions.

In an abstract way, the representation of a problem $P = (a, S, E, F)$ as a space of states consists on:

- A set of states S and an initial state, $a \in S$
- A set E of ordered pairs (x, y), called *transitions*, where x and y are states and y is reachable from x in one step.
- A set F of final states.

Technically, we also need a *cost* mapping, which assigns a cost to each transition (x, y), but we will consider a constant cost and we will omit it.

Given a problem $P = (a, S, E, F)$, we will consider a P system $\Pi = (\Gamma, H, \mu, w_e, w_s, R_1, R_2, R_3, R_1 > R_2, > R_3)$ where

- The alphabet $\Gamma = S \cup \{p_e, r_e \,|\, e \in E\}$
- The set of labels $H = \{u, s\}$
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- The initial multisets $w_u = \{a\}$ and $w_s = \emptyset$.

- Three sets of rules R_1, R_2 and R_3
  - $R_1 = \{[x]_u \to \lambda : x \in F\}$. For each final state we have a dissolution rule which dissolves the membrane u.
  - $R_2 = \{[x \neg p_y \to y\, r_{xy}]_u \,:\, (x,y) \in E\}$. For each transition (x,y), x can be changed by $y\, r_{xy}$ if p_y does not occur in the membrane u, i.e., p_y acts as an inhibitor.
  - $R_3 = \{[y\, r_{xy} \to x\, p_y]_u \,:\, (x,y) \in E\}$. For each transition (x,y) we have a cooperative rule where the multiset $y\, r_{xy}$ is rewritten as $x\, p_y$ in the membrane u.
- Finally, we have an order among the rules. Rules of R_1 have priority over the other rules and rules from R_2 have priority over rules from R_3.

The intuition behind the objects is the following: In each configuration (but in the last one) there is one object from S in the configuration. It represents the current state in the searching process. For each state y, the object p_y is an inhibitor[2] which forbid to visit the state y. Finally, the occurrence of the object r_{xy} represents that the transition (x,y) belongs to the path from the initial state to the current one. We illustrate one computation of these P systems with the following example.

4.1 Example

Let us consider the space of states $P = (a, S, E, F)$ with a the initial state, the set of transitions $E = \{(a,b),(a,c),(b,d),(b,e),(e,f),(c,g)\}$ and the set of final states $F = \{g\}$. Let Π be the P system associated with this space as described above. The initial configuration is $C_0 = [\,[a]_u\,]_s$. Two rules are applicable, both belonging to the set R_2, $r_b \equiv [a \neg p_b \to b\, r_{ab}]_u$ and $r_c \equiv [a \neg p_c \to c\, r_{ac}]_u$. Let us suppose that non-deterministically r_b is chosen. Then we have $C_1 = [\,[b\, r_{ab}]_u\,]_s$. From C_1, three rules are applicable

$$r_d \equiv [b \neg p_d \to d\, r_{bd}]_u \in R_2 \quad r_e \equiv [b \neg p_e \to e\, r_{be}]_u \in R_2 \quad r_{\underline{b}} \equiv [b\, r_{ab} \to a\, p_b]_u \in R_3$$

Since R_2 has priority over R_3, only r_d or r_e can be non-deterministically chosen. We choose r_e and reach $C_2 = [\,[e\, r_{ab}\, r_{be}]_u\,]_s$. Now, only two rules are applicable

$$r_f \equiv [e \neg p_f \to f\, r_{ef}]_u \in R_2 \qquad r_{\underline{e}} \equiv [e\, r_{be} \to b\, p_e]_u \in R_3$$

Since R_2 has priority, r_f is applied and we reach $C_3 \equiv [\,[f\, r_{ab}\, r_{be}\, r_{ef}]_u\,]_s$. From C_3, the unique applicable rule is $r_{\underline{f}} \equiv [f\, r_{ef} \to e\, p_f]_u \in R_3$ and $C_4 \equiv [\,[e\, r_{ab}\, r_{be}\, p_f]_u\,]_s$. Notice than the application of $r_{\underline{f}}$ is an implementation of backtracing. In the configuration C_4, the current state is e and the state f is forbidden. From C_4, only $r_{\underline{e}} \equiv [e\, r_{be} \to b\, p_e]_u \in R_3$ is applicable. The application of this rule is a new step of backtracing and it leads us to the configuration $C_5 \equiv [\,[b\, r_{ab}\, p_e\, p_f]_u\,]_s$. From C_5, two rules are applicable

[2] Notice taht the object p_y is never removed. If the state y can be reached from different paths, then we should add new rules in order to prevent it.

$$r_d \equiv [b\, \neg p_d \rightarrow d\, r_{bd}]_u \in R_2 \qquad r_{\underline{b}} \equiv [b\, r_{ab} \rightarrow a\, p_e]_u \in R_3$$

Notice that the rule $r_e \equiv [b\, \neg p_e \rightarrow e\, r_{be}]_u \in R_2$ is not applicable due to the occurrence of the inhibitor p_e in the membrane u. Since R_2 has priority over R_3, the rule r_d is applied an the configuration $C_6 \equiv [\,[d\, r_{ab}\, r_{bd}\, p_e\, p_f]_u\,]_s$ is reached. From C_6 we only can do backtracing by applying the rule $r_{\underline{d}} \equiv [d\, r_{bd} \rightarrow b\, p_d]_u \in R_3$ and reach $C_6 \equiv [\,[b\, r_{ab}\, p_d\, p_e\, p_f]_u\,]_s$. By applying now $r_{\underline{b}} \equiv [b\, r_{ab} \rightarrow a\, p_b]_u \in R_3$ we obtain $C_7 \equiv [\,[a\, p_b\, p_d\, p_e\, p_f]_u\,]_s$. From C_7 we only can apply $r_c \equiv [a\, \neg p_c \rightarrow c\, r_{ac}]_u \in R_2$ and reach $C_8 \equiv [\,[c\, r_{ac}\, p_b\, p_d\, p_e\, p_f]_u\,]_s$. From C_8 two rules are applicable

$$r_g \equiv [c\, \neg p_g \rightarrow g\, r_{cg}]_u \in R_2 \qquad r_{\underline{c}} \equiv [c\, r_{ac} \rightarrow a\, p_c]_u \in R_3$$

Due to the priority of R_2 over R_3, r_g is applied and we obtain $C_9 \equiv [\,[g\, r_{ac}\, r_{cg}\, p_b\, p_d\, p_e\, p_f]_u\,]_s$. Finally, the applicable rules are

$$r_F \equiv [g]_u \rightarrow \lambda \in R_1 \qquad r_{\underline{g}} \equiv [g\, r_{cg} \rightarrow c\, p_g]_u \in R_3$$

Since R_1 has priority over R_3, the rule r_F is applied and the configuration $C_{10} \equiv [r_{ac}\, r_{cg}\, p_b\, p_d\, p_e\, p_f]_s$. No more rules are applicable and C_{10} is a halting configuration. The objects r_{ac} and r_{cg} determine a path from the initial state to the final one. Notice that the chosen rules in the non-deterministic points are crucial. From C_0 the configuration $C_3^* \equiv [r_{ac}\, r_{cg}]_s$ is reachable in three steps by applying sequentially the rules $r_c \equiv [a\, \neg p_c \rightarrow c\, r_{ac}]_u \in R_2$, $r_g \equiv [c\, \neg p_g \rightarrow g\, r_{cg}]_u \in R_2$ and $r_F \equiv [g]_u \rightarrow \lambda \in R_1$.

5 A New Solution for the N-queens Problem

The first step for designing a new solution for the N-queens problem is to determine the space of states. There are two basic formulations (see [7]). A *complete-state formulation*, which starts with N queens on the board and moves them around and an *incremental formulation*, where the operators augment the state description, starting from the empty state and each action adds a queen to the state. This second formulation reduces drastically the space of states, since a new queen added to the description of a state can be placed only in a non forbidden square. In such way, states and transitions are the following:

- **States:** Arrangements of k queens ($0 \leq k \leq N$), one per column in the leftmost k columns.
- **Transitions** (x, y)**:** The state y is the state x where a new queen is added in the leftmost empty column. Such new queen is not attacked by any other one.

The basic idea of the P system design is to encode the position of a queen as a set of four objects x_i, y_j, u_{i-j} and v_{i+j}, where x_i represents a column and y_j represents a row ($1 \leq i, j \leq N$). The objects u_{i-j} and v_{i+j} represent the ascendant and the descendant diagonals respectively and their subindices are determined by the corresponding column and row i and j. Placing a queen on the chessboard means to choose a square, i.e., a set $\{x_i, y_j, u_{i-j}, v_{i+j}\}$ among the eligible objects and delete them from the corresponding membrane. The choice is recorded. If the final state is reached we finish the process; otherwise we do backtracing and choose other eligible set.

We present a family of P systems which find a solution for the N-queens problem (a P system for each value of N) slightly different from the general one presented in Section 4. We add a new set of rules R^* for cleaning purposes. For each positive integer greater than 2, we consider the P system

$\Pi = (\Gamma, H, \mu, w_e, w_s, R_1, R^*, R_2, R_3, R_1 > R^* > R_2, > R_3)$ where

- The alphabet $\Gamma = \{x_i, y_j, u_{i-j}, v_{i+j}, p_{i,j} \,:\, i, j \in \{1, \dots, N\}\} \cup \{x_{N+1}\}$
- The set of labels $H = \{u, s\}$
- The initial multisets $w_u = \{x_1, y_1, \dots, y_N, u_{1-N}, \dots, u_{N-1}, v_2, \dots, v_{2N}\}$ and $w_s = \emptyset$.
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- Four sets of rules R_1, R^*, R_2 and R_3
  - $R_1 = \{[x_{N+1}]_u \to \lambda : x \in F\}$. In this design, when the object k_N is reached, the membrane u is dissolved and the computation ends.
  - $R^* = \{[p_{i,j} x_{i-1} \to x_{i-1}]_u \,:\, i \in \{2, \dots, N\},\, j \in \{1, \dots, N\}\}$ Just cleaning rules.
  - $R_2 = \{[x_i\, y_j\, u_{i-j}\, v_{i+j}\, \neg p_{i,j} \to x_{i+1}\, r_{i,j}\,]_u \,:\, i, j \in \{1, \dots, N\}\}$ These rules put a new queen on the chessboard by choosing an eligible position.
  - $R_3 = \{[r_{i,j}\, x_{i+1} \to x_i\, y_j\, u_{i-j}\, v_{i+j}\, p_{i,j}]_u\ i, j \in \{1, \dots, N\}\}$. These rules remove one queen form the chessboard and implement the backtracing.
- Finally, the order $R_1 > R^* > R_2, > R_3$ among the sets of rules is settled.

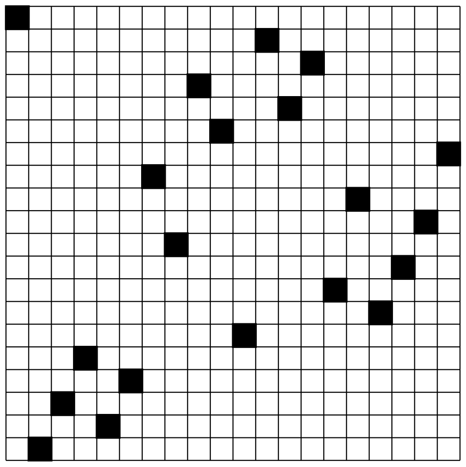
5.1 Hints on the computation

From the objects $\{x_1, \dots, x_N\}$, only x_1 occurs in the initial configuration. This means that the column 1 is already chosen. In order to take the row, one of the N rules $[k_0\, x_1\, y_j\, u_{1-j}\, v_{1+j}\, \neg p_{1,j,0} \to x_2\, r_{1,j,1}\, k_2]_u$ where $j \in \{1, \dots, N\}$ is chosen. The election of this rule determines the square (x_1, y_j) where the first queen is placed. The application of the rule removes the objects corresponding to the column, row ascendant and descendant diagonal lines $x_1\, y_j\, u_{1-j}\, v_{1+j}$ in the chessboard. The associated column, row and diagonals to these objects are no eligible and the new queen will be put in a safe square. The application of the rule produces the object x_2. Next, a rule from the set $[k_1\, x_2\, y_j\, u_{2-j}\, v_{2+j}\, \neg p_{2,j,1} \to x_3\, r_{2,j,2}\, k_3]_u$ is chosen. If the successive choices are right, then the object k_N is reached and the membrane u dissolved. The objects $r_{i,j,r}$ in the membrane s from the halting configuration give us a solution to the problem. If no rules from the set R_2 can be applied, then we apply one rule from R_3. As shown in the general case, such rules implement backtracing and produces objects $p_{i,j,r}$ which act as inhibitors. Before applying rules from R_2 or R_3, the P system tries to apply rules from R_1, which means the halt of the computation, or from R^*, which clean useless inhibitor objects.

5.2 Examples

Figure 3 shows a computation of the P system which solves the four queens problem, where the rules are non deterministically chosen. The subindices of the objects $r_{1,2}r_{2,4}r_{3,1}r_{4,3}$ in the halting configuration give us the found solution. In this case the squares for the four queens are $(1,2),(2,4),(3,1),(4,3)$.

An *ad hoc* CLIPS program has been written based on this design of solution for the N-queens problem based on Membrane Computing techniques. Figure 2 shows a solution for the 20-queens problem found by such computer program.



1-20 2-1 3-3 4-5 5-2 6-4 7-13 8-10 9-17 10- 15
11-6 12-19 13-16 14-18 15-8 16-12 17-7 18-9 19-11 20-14

Fig. 2. A solution for the 20-queens problem

6 Conclusions and Future Work

The purpose of this paper is twofold. On the one hand, to stress the inviability of solutions based on brute force algorithms for intractable problems, even in case of a future implementations. On the other hand, to open a door in Membrane Computing to Artificial Intelligence techniques, which are broadly studied and which can enrich the methodology of the design of P system solutions.

This first approach can be improved in many senses. As pointed out in Section 4, the aim of this paper is not minimalist and probably, searching algorithms can be implemented into P systems by using more simple P system models. The second improvement is associated to the nature of P systems. The design of P systems

Applied rule	*Configuration*
	$C_0 \equiv \left[\left[x_1 y_2 y_3, y_4 u_{-3} \dots u_3 v_2 \dots, v_8 \right]_u \right]_s$
$[x_1 y_1 u_0 v_2 \neg p_{1,1} \rightarrow x_2 r_{1,1}]_u$	$C_1 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3}, u_{-2} u_{-1} \\ u_1 u_2 u_3 v_3 \dots v_8 r_{1,1} \end{array} \right]_u \right]_s$
$[x_2 y_3 u_{-1} v_5 \neg p_{2,3} \rightarrow x_3 r_{2,3}]_u$	$C_2 \equiv \left[\left[\begin{array}{l} x_3 y_2 y_4 u_{-3}, u_{-2} u_1 u_2 u_3 \\ v_3 v_4 v_6 v_7 v_8 r_{1,1} r_{2,3} \end{array} \right]_u \right]_s$
$[r_{2,3} x_3 \rightarrow x_2\, y_3\, u_{-1}\, v_5\, p_{2,3}]_u$	$C_3 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3}, u_{-2} u_{-1} u_1 u_2 \\ u_3 v_3 v_4 v_5 v_6 v_7 v_8 r_{1,1} p_{2,3} \end{array} \right]_u \right]_s$
$[x_2\, y_4\, u_{-2}\, v_6\, \neg p_{2,4} \rightarrow x_3\, r_{2,4}]_u$	$C_4 \equiv \left[\left[\begin{array}{l} x_3 y_2 y_3 u_{-3} u_{-1} u_1 u_2 u_3 \\ v_3 v_4 v_5 v_7 v_8 r_{1,1} r_{2,4} p_{2,3} \end{array} \right]_u \right]_s$
$[x_3 y_2 u_1 v_5 \neg p_{3,2} \rightarrow x_4\, r_{3,2}]_u$	$C_5 \equiv \left[\left[\begin{array}{l} x_4 y_3 u_{-3}, u_{-1} u_2 u_3 v_3 v_4 \\ v_7 v_8 r_{1,1} r_{2,4} r_{3,2} p_{2,3} \end{array} \right]_u \right]_s$
$[r_{3,2} x_4 \rightarrow x_3 y_2 u_1 v_5 p_{3,2}]_u$	$C_6 \equiv \left[\left[\begin{array}{l} x_3 y_2 y_3 u_{-3}, u_{-1} u_1 u_2 u_3 v_3 v_4 \\ v_5 v_7 v_8 r_{1,1} r_{2,4} p_{3,2} p_{2,3} \end{array} \right]_u \right]_s$
$[r_{2,4} x_3 \rightarrow x_2 y_4 u_{-2} v_6 p_{2,4}]_u$	$C_7 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3} u_{-2} u_{-1} \\ u_1 u_2 u_3 v_3 v_4 v_5 v_6 v_7 v_8 \\ r_{1,1} p_{3,2} p_{2,3} p_{2,4} \end{array} \right]_u \right]_s$
$[p_{3,2} x_2 \rightarrow x_2]_u$	$C_8 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3} u_{-2} u_{-1} \\ u_1 u_2 u_3 v_3 v_4 v_5 v_6 v_7 v_8 \\ r_{1,1} p_{2,3} p_{2,4} \end{array} \right]_u \right]_s$
$[r_{1,1} x_2 \rightarrow x_1 y_1 u_0 v_2 p_{1,1}]_u$	$C_9 \equiv \left[\left[\begin{array}{l} x_1 y_1 y_2 y_3 y_4 u_{-3} u_{-2} \\ u_{-1} u_0 u_1 u_2 u_3 v_2 v_3 v_4 v_5 \\ v_6 v_7 v_8 p_{1,1} p_{2,3} p_{2,4} \end{array} \right]_u \right]_s$
$[p_{2,3} x_1 \rightarrow x_1]_u$	$C_{10} \equiv \left[\left[\begin{array}{l} x_1 y_1 y_2 y_3 y_4 u_{-3} u_{-2} \\ u_{-1} u_0 u_1 u_2 u_3 v_2 v_3 v_4 v_5 \\ v_6 v_7 v_8 p_{1,1} p_{2,4} \end{array} \right]_u \right]_s$
$[p_{2,4} x_1 \rightarrow x_1]_u$	$C_{11} \equiv \left[\left[\begin{array}{l} x_1 y_1 y_2 y_3 y_4 u_{-3} u_{-2} \\ u_{-1} u_0 u_1 u_2 u_3 v_2 v_3 v_4 v_5 \\ v_6 v_7 v_8 p_{1,1} \end{array} \right]_u \right]_s$
$[x_1 y_2 u_{-1} v_3 \neg p_{1,2} \rightarrow x_2 r_{1,2}]_u$	$C_{12} \equiv \left[\left[\begin{array}{l} x_2 y_1 y_3 y_4 u_{-3} u_{-2} u_0 u_1 u_2 \\ u_3 v_2 v_4 v_5 v_6 v_7 v_8 p_{1,1} r_{1,2} \end{array} \right]_u \right]_s$
$[x_2 y_4 u_{-2} v_6 \neg p_{2,4} \rightarrow x_3 r_{2,4}]_u$	$C_{13} \equiv \left[\left[\begin{array}{l} x_3 y_1 y_3 u_{-3} u_0 u_1 u_2 u_3 \\ v_2 v_4 v_5 v_7 v_8 p_{1,1} r_{1,2} r_{2,4} \end{array} \right]_u \right]_s$
$[x_3 y_1 u_2 v_4 \neg p_{3,1} \rightarrow x_4 r_{3,1}]_u$	$C_{14} \equiv \left[\left[\begin{array}{l} x_4 y_3 u_{-3} u_0 u_1 u_3 v_2 v_5 v_7 v_8 \\ p_{1,1,0} r_{1,2} r_{2,4} r_{3,1} \end{array} \right]_u \right]_s$
$[x_4 y_3 u_1 v_7 \neg p_{4,3} \rightarrow x_5 r_{4,3}]_u$	$C_{15} \equiv \left[\left[\begin{array}{l} x_5 u_{-3} u_0 u_3 v_2 v_5 v_8 \\ p_{1,1} r_{1,2} r_{2,4} r_{3,1} r_{4,3} \end{array} \right]_u \right]_s$
$[x_5] \rightarrow \lambda$	$C_{16} \equiv \left[\begin{array}{l} u_{-3} u_0 u_3 v_2 v_5 v_8 \\ p_{1,1} r_{1,2} r_{2,4} r_{3,1} r_{4,3} \end{array} \right]_s$

Fig. 3. Example of computation

which computes searching is too close to the classical sequential algorithm. In fact, although the presented P system family uses non-determinism in the choice of the rules, it does not explore the intrinsic parallelism of P systems. The next step in this way is to design algorithms which uses a limited form of parallelism where several rules can be applied simultaneously, but controlling the exponential explosion of brute force algorithms. The current hardware based on Compute Unified Device Architecture [4] from Nvidia can be a clue for these new generation of algorithms.

Acknowledgements

The authors acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos–Núñez. A P-Lingua Programming Environment for Membrane Computing. *Lecutre Notes in Computer Science*, **5391**, (2009), 187-203.
2. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos–Núñez, F.J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* **83**(7), (2006) 593 - 611.
3. M.A. Gutiérrez-Naranjo, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Solving the N-Queens Puzzle with P Systems. Seventh Brainstorming Week on Membrane Computing. Vol I. R. Gutiérrez-Escudero, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos–Núñez (Eds.). Fénix Editora, Sevilla (2009) 199-210.
4. M.A. Martínez-del-Amor, I. Pérez-Hurtado, Mario J. Pérez-Jiménez, J.M. Cecilia, G. Guerrero, J.M. García. Simulation of Recognizer P Systems by Using Many-core GPUs. Seventh Brainstorming Week on Membrane Computing. Vol II. M.A. Martínez-del-Amor, E.F. Orejuela-Pinedo, Gh. Păun, I. Pérez-Hurtado, A. Riscos–Núñz (Eds.). Fénix Editora, Sevilla (2009) 45-57.
5. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrinini. A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil (Eds.). Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, Computer and Automaton Research Institute of the Hungarian Academy of Sciences (2003), pp.:284-294.
6. M.J. Pérez-Jiménez. A Computational Complexity Theory in Membrane Computing. *Lecture Notes in Computer Science*, **5957** (2010), 125-148.
7. S. Russell, P. Norvig. Artificial Intelligence. A Modern Approach. Second Edition. Pearson Education, Inc. 2003.
8. Wikipedia. `http://en.wikipedia.org/wiki/Orders_of_magnitude_(mass)`

Plain Talk about Systems Complicatedness

Jozef Kelemen

Institute of Computer Science, Silesian University
Opava, Czech Republic
and VSM College of Management, Bratislava, Slovakia
kel10um@axpsu.fpf.slu.cz

1 Introduction

This extremely informal contribution sketches an extremely informal proposal of classification of systems according their (up to now only intuitively) comprehended *complicatedness* by providing few examples from the fields of (theoretical) computer science, (computer) arts, and economics. The intuitive comprehension provides, however, at least in certain extent, an extension of theoretically well-founded and deeply studied classification of computing systems and their behavior (computation, algorithms) according different complexity measures into complexity classes of systems and behaviors with the same complexity as known in the traditional theoretical computer science. Almost all of our professional considerations are up to now traditional in the sense that almost all of us try to cover all of the appealing objects (languages, molecules, membrane structures), and phenomena (sentence generations, DNA mutations, cells functioning) into the traditional, of course in many situations very well-working, paradigm of the *Turing computability*, and of the spectrum of formal models performing these type of computation.

Our aim is to extend the typology of systems by some intuitive classes of systems with more or less similar level of *complicatedness* in order to include to the potential formalistic debate also systems inspired by some advances in artificial intelligence, artificial life, cognitive science and similar disciplines despite of this notion has - at least up to now - not a very clear status in the hardcore theories of computation.

First of all, it seems to by reasonable to specify at least intuitively what we will in this plain talk understood as a *system*. We will concentrate to systems which produce some symbolic behaviors or structures on the base of transforming another (sensed) symbols or symbol structures into the form of their output structures or their behavioral units. More or less *autonomous agents* as described e.g. in (Kelemen, 2006) are examples of such systems. The next sections will provide a couple of examples.

We note that from our point of view also collections of systems working together in order to generate a behavior (e.g. multi-agent systems, decentralized systems, societies of systems) may be considered as unique systems.

Our second duty is the - at least intuitive - a specification of the meaning of *complicatedness*. By complicatedness we will mean a structural property of a system which provides a base for producing the behavior of this system. If a behavior is given, then "What is the less complicated system which is able to perform this behavior?" seems to be one among the most appealing question related to systems complicatedness.

2 Simple Systems

Simple systems are, according our intuition, systems having a specific property - if we have several such systems, then behavior of the society composed from such systems will be in certain meaning only the simple result of their individual behavior.

More formally and more generally speaking from the point of view of their behavior are simple systems closed under set-algebraic operation like union, intersection, Kleene's * operation, and set complementation operation. From such a perspective, some of the formal grammars - understood as generators of behaviors in the form of sets of strings of symbols (languages) are simple systems: Having two context-free grammars, the behavior which results from the union of their individual behaviors (context-free languages generate by them) is again a behavior (a context-free language) which can be generated by another corresponding context-free grammar. Similar is the situation with other models, too: If we have, for instance, two finite automata, and we form the set-theoretic union (or we use some other suitable operation) of their behaviors (the union of two regular languages accepted by them, or some other operation over these two languages) we receive a regular language again, and we are able to construct a finite automaton which will accept the resulting regular language.

From the position of the theoretical computer science we may state, that simple systems are all theoretical models related to computation (automata, machines, grammars) the class of languages corresponding to which are closed under the traditional (above mentioned) set-theoretic (and also to the other traditional ones - concatenation, and reversal) operations usually studied in theoretical computer science. More details on these models and their closures under set-theoretic (and some other) operations are included into each course-book on theoretical computer science; let us mention e.g. (Hopcroft, Ullman, 1969, Chapter 9). The infinitely large classes of regular, context-free, context-sensitive, and recursively enumerable languages (sets) have this property with respect the operation of union, concatenation, intersection, and reversal; see (Hopcroft, Ullman, 1969, Theorem 9.1).

There are several possibilities of how to define different *complexity measures* which reflects in a very formal, theoretic level some of the characteristics of simple systems, and how to use the differences induced by these measures in order to stratify simple systems according these measures into different complexity classes, to relate these classes, to study the possibility of partial reduction of one to another, etc.; more details provides e.g. (Hromkovic, 1997).

Another possibilities of how to define different classes of languages using some biologically well-inspired models of computing devices can be found e.g. in (Paun, Rozenberg, Salomaa, 1998) for computational interpretations of some of important biochemical processes appearing between nucleic acid macromolecules, and in (Paun, 2002) for the case of computing motivated by biochemical and biophysical processes appearing in (organic) membrane systems like cells, for instance.

3 Systems with Emerging Behavior

The traditional and most widely used informal definition of *emergence* is formulated in (Holland, 1998, pp. 121-122): Emergence is "... a product of coupled, context-dependent interactions. Technically these interactions, and the resulting system, are nonlinear: The behavior of the overall system cannot be obtained by summing the behaviors of its constituent parts... However, we can reduce the behavior of the whole to the lawful behavior of its parts, if we take nonlinear interactions into account".

In connection with the phenomenon of emergence, another phenomenon appeared very interesting form the computational point of view - the notion of emergent computation. The premise of emergent computation is - according (Forrest, 1991, p. 1) - that interesting and useful computational systems can be constructed by exploiting interactions among primitive components, and further, that for some kinds of problems (e.g. modeling intelligent behavior) it may be the only feasible method. The formal study of such processes and the systems behind them is in the focus of the professional attention up to now, and might be interesting to reflect it not only in experiments in the field of artificial intelligence and artificial life, but also in the context of theories of formal symbolic behavior generators.

Systems with emerging behavior are in fact multi-agent systems because they are set up from a number of individually behaving component systems. Component systems have their own behaviors, and they have also some possibilities to communicate in some indirect ways, sharing the common "environment", for instance, say, e.g., rewriting symbols in a shared string. Good candidates for become to be *systems with emerging behavior* are grammar systems as presented in (Csuhaj-Varju et al., 1994), or *eco-grammar systems* (Csuhaj-Varju et al., 1997).

Consider now component systems to be simple systems from some complexity classes, and consider the behavior of the whole system composed from the component systems now. We may recognize two possibilities:

1. the systems will produce behavior from one of the complexity classes of the component systems, or

2. the system will produce a behavior from another complexity class. In the second case the behavior of the system is emergent, it emerges from the behaviors of the component systems, and the systems will be called system with emerging behavior.

The emergent behavior of such systems satisfies the *emergence test* formulated in (Roland et al., 1999) consisting in the following three basic testing steps:

a) *Design.* The designer designs the systems by describing *local* interactions between components in a language L1.
b) *Observation.* The observer describes *global* behaviors of the running system using a language L2.
c) *Surprise.* The language of design L1 and the language of observation L2 are distinct, and the causal link between the elementary interactions programmed in L1 and the observations observed in L2 are *non-obvious.*

A suitable example of systems with emerging behavior are variants of *grammar systems*, c.f. (Csuhaj-Varju et al., 1994), called *colonies* (Kelemen, Kelemenova, 1992). In the case of colonies finite sets of regular grammars cooperating as members of a grammar system are able to generate the members of all of the family of context-free languages; the relation of this phenomenon to the emergence is discussed in (Kelemen, 2004). Some remarks on a possibility how to attack the formal treatment of the problem of emergence from the positions of the traditional formal language theory and the theory including the theory of abstract families of languages into considerations, is presented in (Freund, Kelemen, Paun, 2003).

4 Hyper-Computing Systems

Hyper-computing systems are, very roughly speaking, systems, which go by their computing potentials in certain senses beyond the limits of traditional Turing-computation. Burgin and Klinger (2004) described the relevant opinions, and in the special issue of TCS Journal in which the just mentioned article is published and which is edited by them, collects a couple of other interesting opinions.

In (Stannett, 2004) the problem of hyper-computation is connected in an elegant way by the Turing machine and the Church-Turing thesis by making explicit the following three points:

1. Computation in a Turing machine is in fact a controlled manipulation of configurations, where each configuration encodes a finite amount of information as a state, a finite amount of information as memory, and a finite amount of information as program.
2. Turing machines control structure is constrained both by the current configuration of it, and by the requirement that only one program instruction is executed at a time.

Then the Church-Turing thesis expresses the conviction that any "cosmetic changes" in the architecture of the Turing machine have no principal influence to its the computational power (may be they have some influence the traditionally understood complexity requirements of performed computation, but what is not computable by a Turing machines remains not computable by other machines, too).

Stannett in the above mentioned article concludes with the statement, that there are only four obvious ways of modifications of the Turing machine: the temporal structure of computation, the information contents of memory, the information content of programs, and the information content of states. Then he provides the list of publications which attacked the problem from the mentioned obvious ways.

Let us to provide an example of such systems producing non recursive behavior which is not mentioned in (Stannett, 2004). In connection with another variant of grammar systems - with so called eco-grammar systems (Csuhaj-Varju et al., 1997) - some observations concerning the hyper-computing potentials of this model is provided in (Watjen, 2003). Roughly and informally speaking, Watjen in his just mentioned article proved that if into an eco-grammar system which uses teams of components for generating strings of symbols, a non-recursive function is included which defines the number of components in teams for each step of the derivation process, then such an eco-grammar system is able to generate non-recursive languages. Let us mention marginally, that the pure randomness comprehended as a function (in the above case prescribing teams to derivation steps) seems to be, at least intuitively, non-recursive (in the opposite case it is not random), and that perhaps in the real, non idealized situations the randomness play very crucial role in real physically embodied systems behaviors.

5 Creative Systems

To *create* is usually used for denoting the ability to cause something to come to existence, bring into being, originate something. Creativity is then usually considered as the act of creation, so as a mental and social process involving the generation of new concepts or new associations between the existing concepts, the ability to finding "new ways to look at things"; cf. (Minsky, 1986, p. 134). Two principal attributes are usually required with respect to creative combinations of concepts - the *originality* of the concept, and its *appropriateness.*

In order to decision about the appropriateness, in all of usual cases an anthropocentric test (in certain extent similar to the Turing test known in AI) is used - a test of the appropriateness in a given cultural context of a given human society. The culture of the particular human society dictates what is required and what is (at least marginally) acceptable. This is the important and inevitable outer anthropocentric determination of the inner individual creativity of each human mind or some artificial information processing systems, too.

During the history of the development of computers use, and during the experiments with computational models of mental processes in the field of artificial intelligence and cognitive science numerous systems have been developed which are based on some hypotheses concerning the human mental activities. It have been developed also a challenging area of agents, and multi-agent systems, the fields which are focused to the understanding, construction, and practical use of more and more autonomous computer-based systems - agents as presented e.g. in (d'Inverno, Luck, 2001) -, and societies in which agents may interact - multi-agent systems; see e.g. (Ferber, 1999).

The existence of (societies of) agents opens another, from some perspective - if we consider human societies as a kind of multi-agent systems, and the interactions of agents inside multi-agent systems as some level of the existence of culture in these systems - a more general position for testing the appropriateness of results of creativity.

Some artificially designed systems produce behaviors which, in the case that a human being is considered as the system of this type, are called creative. The author of this contribution does not know any theoretical, formal approach to the study of systems of this type. However, there are some successfully working systems of this type, and now we will mention examples of the two types of creative systems in certain details on the base of (Kelemen, 2009).

In the field of artificial intelligence, numerous systems have been developed which belong to the broad family of the so called *goal-driven systems*. The basic idea common for all of such systems consists in comparing some representation of an actual situation of a given problem with (representation of) a given desired situation of it. The comparison results in an ordered set of formally defined and represented *differences*. The differences are then, step by step, *reduced* using some formally defined *operators* in order to reduce the number of differences between the existing and the desired state of the problem. Operators and differences are connected with respect the ability of the given operator to reduce the related with it difference(s).

The principle was successfully applied e.g. in the famous *General Problem Solver* (GPS) developed by H. A. Simon, A. Newell, and C. J. Shaw during the end of fifties and beginning of sixties of the past century; for more details see e.g. (Ernst, Newell, 1969).

A crucial point in the GPS which makes it relevant for our discussion is that from simple concepts (operators) it constructs autonomously (without any human assistance) a sequence of operators, a more complicated concept, which represents the solution of the problem given at the beginning as an input to the system. This sequence, if GPS is successful in solving the given problem, transforms the starting situation describing the problem (the well-known tower-of-Hanoi or some similar problems), into the situation which represents its solution. In this sense GPS represents a creative system. However, its creativity is fundamentally based on the definitions of operators, differences, situation descriptions, table of operator-differences connections, etc. provided by its human users. So, the success of GPS in solving problems depends on the quality of these human-defined components.

Art is another field in which the agent paradigm works well, and creativity exhibits its potentials very clearly. Perhaps the most popular among the computer-based art-producing machines is the system called AARON. It began - according its author Harold Cohen (Cohen, 1995) - its existence some time in the mid-seventies of the past century; see e.g. (Cohen, 1973). The earliest versions of the system used some perceptual primitives only for producing (drawing) images. It has the ability to differentiate between figure and the ground, to differentiate between open forms and closed forms, and to differentiate between insideness and outsideness. Moreover, it has the capability to perform various simple manipulations on the structures it produced. Time-to-time, this more or less randomly executed manipulation on primitive line-structures resulted in figures having in certain sense figurative contents, like "human face" represented for instance by the expression as follows: human-face IS (INSIDEclosed-form (UPPER-POSITION(closed-form, closed-form) CENTRAL-POSITION(open form), DOWN(closed-form)))

This symbolic representation may look in corresponding free-hand drawing representation like in Fig. 1 (a). A more complicated picture - an original drawing made by AARON Fig. 1 (b) - of the "human body" may be represented in similar symbolic way. Of course, the human face can be then sophisticated also to more and more complicated pictures, e.g. as the one in Fig. 1 (c), produced with a more sophisticated version of AARON. Realize that what the human-face expresses, is the definition of the human face for AARON.

We decided, associating the human-face with its formal expression that the randomly scrawled lines remind the line-drawing of a human face in our minds. Now we are able to instruct AARON to draw a new human face. But the result will be not the same, as in the previous case, the same will be only the structure expressed by the formal definition of what we consider to be a human face. In such a way we can produce more and more concepts and instruct AARON to draw more and more complicated drawings, each with certain degree of "freedom" of AARON's drawing. For generating Fig. 2 the requirement was a picture of a botanical garden with seven human beings (five distressed women and two men) inside it.

AARON is - from the point of view of knowledge processing technologies - in fact a *rule-based knowledge-system* in the usual meaning used e.g. in (Stefik, 1995). Without going into the technical details of the construction (programming) of it, we may conclude, in concordance with conclusions made in (Cohen, 1995), that AARON constitutes an existence proof of the power of machines to do some of the things traditionally connected with human thought and his creativity. "If what AARON is making is not art, what is it exactly, and in what ways, other than its origin, does it differ from the "real thing"? If it is not thinking - and let us to be more explicit: creative thinking -, what exactly is it doing?" Cohen asks in last strokes of (Cohen, 1995).

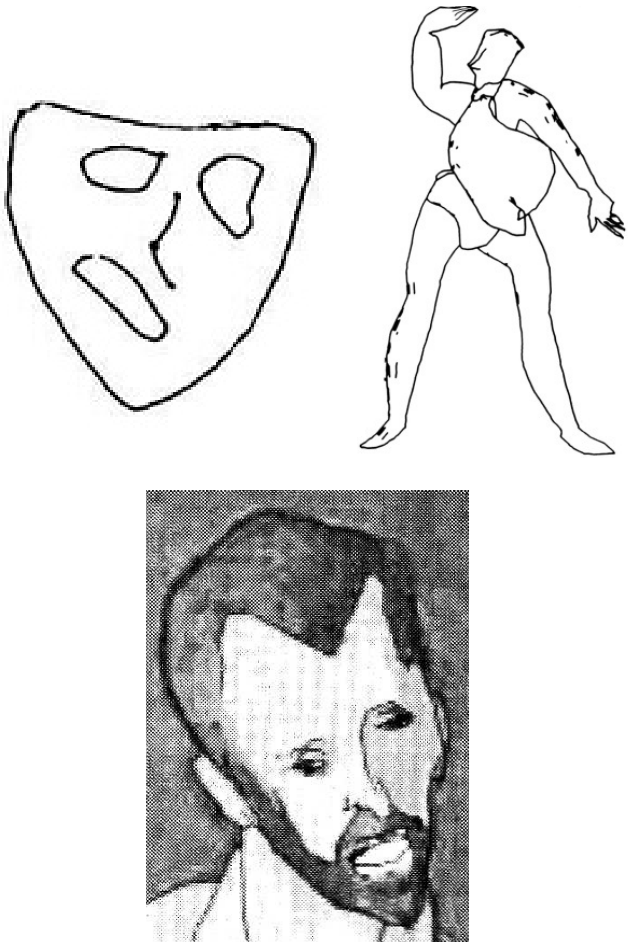


Fig. 1. (Parts of) pictures. The first drown in AARON style; the second produce by AARON, and taken from (McCorduck, 1990, p. 105), the third one produced by AARON, and taken from (Kurzweil, 1999, p. 167).

6 Man-Machine Systems

The emergent nature of some creative phenomena appearing in complicated systems - like the *man-machine societies* are - we may also test using the so called test of emergence proposed in (Ronald et al., 1999). We provide here an example of such a system from the field of the interactive art discussed in more details and with more examples in (Horakova, Kelemen, 2010).

The first interactive piece we mention is based on some ideas from the experiments executed in the field of artificial life, and is created by Christa Sommerer and Laurent Mignonneau. The project named A-Volve has been presented first in

Fig. 2. Five distressed women and two men (one hiding, upper right). Taken from (McCorduck, 1991, p. 135)

1994, and developed it during few next years. The audience of the A-Volve session has possibilities to design artificially living creatures and to provide for them different modes of their behavior (aggression, snuggles, curiosity, caution, friendship, etc.). In the virtual world behind of the screen a society of such creatures mutually interact, and behaves according the habit of the members, and have been influented also if the visitors touched the screen; see Fig. 2. More details on the A-Volve can be found in (Whilelaw, 2004).

Let us now to analyze the project from the position of the *test of emergence*: *Design.* The language L1 is the language in which the system is implemented, so, a purely technical computer programming device used with a specific intention to provide a usable, user-friendly software product for well-specified purposes. *Observer.* The previous language substantially differs, of course, form the language L2 in which the audience - the observers - interacts with the systems A-Volve. This language contains tools for defining new creatures, and contains also kinds of gestures for interacting with the creatures through touching the screen. *Surprise.* The surprise follows then from the observation of the new created creatures as members of he existing community behind the screen, and from the direct interaction with the creatures through touching the screen. This is the reason why we propose that the real artistic creativity emerges in the case of the systems A-Volve from interaction of human being with the machine.FFF

Fig. 3. Interaction with the A-Volve. Photo Ch. Sommerer and L. Mignonneau, 1994.

7 Conclusion - What About to Study Reflexive Systems?

The emergence of creativity mentioned in the previous two chapters can be discussed also in the broader context of a special kind of systems derived from study of some specificity of economic systems and their behavior, and called by George Soros as *reflexive systems* (Soros, 1994). For explaining what kind of systems we have in mind we borrow an example from (Soros, 1994, p. 42).

Let us suppose that active agents belonging to a given system work according two functions. The first function, say f, is defined on the situations appearing on the system. We will call it the cognitive function, because the participants - the agents - effort to understand the system depends on perception of the systems. More formally (but not in a pure formalistic manner) we have $y = f(x)$. The second function defines the participants' participation on the changes of the system. This participation is supposed to be rationalistic, so is based on the understanding of the system and changes the situation inside of the system. We express this dependence of the behavior on the understanding by the function $x = g(y)$ and call this function, according Soros proposal as the participating function. So, as the result we have: $y = f(x)$, and $x = g(y)$, what gives $y = f(g(y))$, and $x = g(f(x))$. This is, roughly speaking, and not in a very well formulated way, the fundamental property of the reflexive systems.

An example: Suppose that a globally influential group of economic experts start to speak and write on a good functioning bank in a highly critical tone and start to hesitate with respect of its economic future. What will happen? This bank will quickly go to problems and in the more wrong case to bankruptcy. More generally speaking, in certain situations, in certain types of systems the observer's observations change the observed object. So, "objective" observations are not possible in this type of systems. This type of systems are reflexive.

The development of a methodology for systematic study of such systems might perhaps start form experimentations with artificially created societies of simple economic agents as presented in a very impressive way in (Epstein, Axtell, 1996). The experiments prove the way how some simple economic laws, e.g. the famous *Pareto law*, emerges - laws originally formulated on the base of observation of the behavior oh human economic societies - in very simple societies with some basics of economic behavior of their members, and how many other interesting economic and social situations and processes can be observed and experimentally tested in the specific test-bed of the multi-agent system the author used in experiments collected and analyzed in the above mentioned book .

What was presented concerning the creativity is another illustration of the behavior of reflexive systems. Thanks to the reflexivity of the human society the new creations are first surprising, but then become to be accepted, so, they become appropriate for the society in which they have first the attribute of innovations (technical innovations, artistic innovations, fashion innovations, etc.).

However, the study of reflexive systems are up to now and according the author best knowledge, out of the scope of interest of theoretical computer scientists, despite of the fact, that reflexivity is perhaps the property of majority of the complicated information processing systems (like human brains, computer networks, man-machine systems and societies, etc.). What about to look for the much more suitable formal frameworks in order to make first steps towards formal understanding of this type of systems from computationalist positions similarly as we do that with the simplified models of real computing engines in the field of traditional computer science?

References

1. Burgin, M., Klinger, A.: Three aspects of super-recursive algorithms and hypercomputation or finding black swans. *Theoretical Computer Science* 317 (2004) 1-11
2. Cohen, H.: Parallel to perception - some notes on the problem of machine-generated art. *Computer Studies* 4, No. 3/4 (1973)
3. Cohen, H.: The further exploits of AARON, painter. *SEHR, Stanford Electronic Humanities Review* 4. No. 2 (1995)
4. Csuhaj-Varju, E., Dassow, J., Kelemen, J., Paun, Gh.: *Grammar Systems - A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach, Yverdon, 1994

5. Csuhaj-Varju, E., Kelemen, J., Kelemenova, A., Paun, Gh.: Eco-grammar systems - a grammatical framework for studying life-like interactions. *Artificial Life* 3 (1997) 1-28
6. D'Inverno, M., Luck, M.: *Understanding Agent Systems.* Springer, Berlin, 2001
7. Epstein, J. M., Axtell, R.: *Growing Artificial Societies.* The MIT Press, Cambridge, Mass., 1996
8. Ernst, G.; Newell, A.: *GPS - A Case Study in Generality and Problem Solving.* Academic Press, New York, 1969
9. Ferber, J.: *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence.* Addison Wesley, New York, 1999
10. Freund, R., Kelemen, J., Paun, G.: A note on emergence in multi-agent string processing systems. *Computing and Informatics* 22 (2003) 623-637
11. Forrest, S. (Ed.): *Emergent Computation.* The MIT Press, Cambridge, Mass., 1991
12. Holland, J.: Emergence. Addison-Wesley, Reading, Mass., 1998
13. Hopcroft, J. E., Ullman, J. D.: *Formal Languages and their Relation to Automata.* Addison-Wesley, Reading, Mass., 1969
14. Horakova, J., Kelemen, J.: On the emergence of creativity in man-machine systems. In: *Proc. 14th IEEE International Conference on Intelligent Engineering Systems, INES 2010*, Las Palmas, Gran Canaria, May 2010 (accepted)
15. Hromkovic, J.: *Communication Complexity and Parallel Computing.* Springer, Berlin, 1997
16. Kelemen, J.: Miracles, colonies, and emergence. In: *Formal Languages and Applications* (C. Martin-Vide et al., eds.). Springer, Berlin, 2004, pp. 323-334
17. Kelemen, J.: Agents from functional-computational perspective. *Acta Polytechnica Hungarica* 3 (2006) 37-54
18. Kelemen, J.: A note on agents, societies, and creativity. In: *Proc. 7th IEEE International Conference on Computational Cybernetics, ICCC 2009* (A. Szakal, Ed.). IEEE CD Publication (IEEE Catalog Number CFP09575-CDR) 2009, pp. 81-84.
19. Kelemen, J., Kelemenova , A.: A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems* 23 (1992) 621-633
20. Kurzweil, R.: *The Age of Spiritual Machines.* Penguin Books, New York, 1999
21. McCorduck, P.: *Aaron's Code - Meta-Art, Artificial Intelligence, and the Work of Harold Cohen.* Freeman and Co., New York, 1991
22. Minsky. M.: *The Society of Mind.* Simon & Schuster, New York, 1986
23. Paun, G.: *Membrane Computing - An Introduction.* Springer, Berlin, 2002
24. Paun, G., Rozenberg, G., Salomaa, A.: *DNA Computing.* Springer, Berlin, 1998
25. Roland, E. M. A., Sipper, M., Capcarrere, M. S.: Design, observation, surprise! A test of emergence. *Artificial Life* 5 (1999) 225-239
26. Soros, G.: *The Alchemy of Finance.* John Wiley and Sons, New York, 1994
27. Stannett, M.: Hypercomputational models. In: *Alan Turing - Life and Legacy of a Great Thinker* (Teuscher, Ch., Ed.). Springer, 2004, pp. 135-157
28. Stefik, M.: *Introduction to Knowledge Systems.* Morgan Kaufmann, San Francisco, Cal., 1995
29. Watjen, D.: Function-dependent teams in eco-grammar systems. *Theoretical Computer Science* 306 (2003) 39-53
30. Whitelaw, M.: *Metacreation - Art and Artificial Life.* The MIT Press, Cambridge, Mass., 2004

Model Checking Based Test Generation from P Systems Using P-Lingua

Raluca Lefticaru[1], Florentin Ipate[1], Marian Gheorghe[1,2]

[1] University of Pitesti, Department of Computer Science
Str Targu din Vale 1, 110040 Pitesti, Romania
`raluca.lefticaru@gmail.com`, `florentin.ipate@ifsoft.ro`

[2] University of Sheffield, Department of Computer Science
Regent Court, Portobello Street, Sheffield S1 4DP, UK
`M.Gheorghe@dcs.shef.ac.uk`

Summary. This paper presents an approach for P system testing, that uses model-checking for automatic test generation and P-Lingua as specification language. This approach is based on a transformation of the transitional, non-deterministic, cell-like P system into a Kripke structure, which is further used for test generation, by adding convenient temporal logic specifications. This paper extends our previous work in this field to multi-membrane, transitional P system, having cooperative rules, communication between membranes and membrane dissolution. A tool, which takes as input a P system specified in P-Lingua and translates it into the language accepted by the model checker NuSMV was developed and used for test case generation. Some hints regarding the automatic test generation using NuSMV and P-Lingua are also given.

1 Introduction

Membrane computing is a branch of natural computing, which investigates parallel computing models, inspired by the structure of the living cell, called *P systems*. These computational models, were introduced by Gheorghe Păun in 1998, in its seminal research report, further published as journal paper [19]. Membrane computing has known a fast growth in the last years: many variants of P systems have been proposed and results concerning their computational power and universality have been obtained. A recent handbook summarizes the most important developments in this field [21]. For all these P system variants, different implementations and simulators have been developed and consequently it appears the necessity of testing these implementations.

A first approach on testing P systems focuses on cell-like models and proposes some coverage criteria [12], which are empirically evaluated in [16]. Automatic test generation for P systems using model-checking is proposed in [15].

Given a model of a system, *model checking* [6] is a formal verification technique that explores the entire state space and decides whether this model meets a given property, expressed in temporal logic. If the property does not hold, then a counterexample is returned. This capability of model checkers to construct counterexamples provides a way to build test sets. Fraser et al. present in a recent and comprehensive survey [10] the results obtained over the last decade in software testing using model checkers.

One of the approaches used to obtain a test suite using model checking follows the steps [10]:

1. A *test purpose* is defined, describing the expected features of the test case, for example: reaching a certain state s in the model, covering a transition t, traversing a sequence of states, getting a certain value *val* of a variable x, etc.
2. These features are further specified as temporal logic properties and then converted by negation into *never-claim* conditions, or *trap properties*, such as: `G !(state = s)`, expressing that the system will never reach state s, or `G !(x = val)`, expressing that the value *val* is never taken (x is always different from *val*).
3. The model checker will verify whether the *never-claim* or *trap property* holds. If the property is false, it returns a counterexample that gives the exact path in the model that reaches state s or sets the system variable x to *val*. The counterexample will provide all the information needed to extract the test case. If the property is true, then it is impossible to build a test case satisfying the given purpose.

Regarding P system testing, one intuitive test criterion is *rule coverage*, that specifies that the test set should contain test cases which cover every rule, i.e. for each rule there exists a test case, describing a computation which involves that rule. More powerful test sets can be computed by considering the *context-dependent rule coverage* criterion. This considers coverage of rules in the context defined by other rules.

An approach on building test cases for P systems using model checking was proposed in [15]. It transforms the P system specification into a Kripke structure, then properties regarding the coverage criteria are expressed in LTL (Linear Temporal Logic) and added to the NuSMV specification. This paper extends the work from [15] in the following aspects:

- It employs the P-Lingua framework [11], to specify the P system and verify its syntactic correctness.
- It uses multi-membrane P systems, having cooperative and communication rules between membranes (the approach presented in [15] treats only one-membrane P systems, with cooperative rules).
- A transformation of P systems with *membrane dissolution* into the SMV (Symbolic Model Verifier) language is proposed.
- *Bounded model checking* is used to obtain the shortest counterexamples. This is useful in practice, to obtain a reduced test suite.
- The paper shows how other properties can be verified against the transformed model, to find possible faults in the P system.

2 Background

In the rest of the paper, we will use the following notations: V^* for the set of all strings over the alphabet $V = \{a_1, ..., a_p\}$ and λ to denote the empty string. For a string $u \in V^*$, $|u|_{a_i}$ denotes the number of a_i occurrences in u. Each string u has an associated vector of non-negative integers $(|u|_{a_1}, ..., |u|_{a_p})$. This is denoted by $\Psi_V(u)$.

2.1 P systems

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. Each region has associated a finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string $u \in V^*$, where the order is not considered, or by $\Psi_V(u)$. The following definition refers to one of the many variants of P systems, namely cell-like P systems, which uses transformation and communication rules [20]. We will call these processing rules. Since now onwards we will call this model P system.

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

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

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

We say that a rule is *cooperative* if it has at least two objects in its left hand side, e.g. $ab \rightarrow (c, in)(d, out)$. Otherwise, the rule is *non-cooperative*, e.g. $a \rightarrow (c, in)(d, out)$. The rules can also have the form $u \rightarrow v\delta$, where δ denotes the action of *membrane dissolution*: if the rule is applied, then the corresponding membrane disappears and its contents, object and membranes alike, are left free in the surrounding membrane; the rules of the dissolved membrane disappear with the membrane. The skin membrane is never dissolved. For further details regarding membrane computing, please refer to [20].

2.2 P-Lingua

P-Lingua is a programming language for membrane computing [8], developed by members of the Research Group on Natural Computing, at the University of Seville. It is developed as a free software framework for cell-like P systems and can be downloaded from http://www.p-lingua.org. Its main component is a Java library, `pLinguaCore`, that accepts as input text files (either in XML or in P-Lingua format) describing the P system model [11].

The library includes several built-in simulators for each supported model. P-Lingua 2.0 was designed for cell-like P systems and contains simulators for the following types of P systems: active membrane with division/creations rules, transition, symport/antiport, stochastic and probabilistic P systems. P-Lingua 2.1 (actual version) was extended for tissue P systems with symport/antiport rules and cell division [17].

The P-Lingua software package contains the `pLinguaCore` library and a user interface called `pLinguaPlugin`. It was used in several research papers, e.g. to solve a SAT problem using a family of P systems [8], to describe and simulate ecosystems by means of P systems [11].

A specification in P-Lingua of the P system $\Pi = (V, \mu, w_1, w_2, R_1, R_2)$, $V = \{s, a, b, c\}$, $\mu = [_1[_2]_2]_1$, $w_1 = s$, $w_2 = \lambda$, $R_1 = \{r_1 : s \rightarrow sa(b, in); r_2 : s \rightarrow ab; r3 : b \rightarrow a; r_4 : a \rightarrow c\}$, $R_2 = \{r_5 : b \rightarrow bc, r_6 : b \rightarrow c\}$ is given in Fig. 1 and can be saved in a specific file, with the `.pli` extension.

2.3 Kripke structures

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

Usually, the Kripke structure representation of a system results by giving values to every variable in each configuration of the system. Suppose $var_1, \ldots, var_n$ are the system variables, Val_i denotes the set of values for var_i and val_i is a value from Val_i, $1 \leq i \leq n$. Then the states of the system are $S = \{(val_1, \ldots, val_n) \mid val_1 \in Val_1, \ldots, val_n \in Val_n\}$, and the set of atomic predicates are $AP = \{(var_i = val_i) \mid 1 \leq i \leq n, val_i \in Val_i\}$. Naturally, L will map each state (given by the values of variables) onto the corresponding set of atomic propositions. For convenience, in

```
@model<transition>
def main()
{
      /* Initial configuration */
 @mu = [[]'2]'1;
      /* Initial multisets */
 @ms(1) = s;
 @ms(2) = #;
      /* Rules */
 [s []'2]'1 --> [s,a [b]'2]'1;
 [s --> a,b]'1;
 [b --> a]'1;
 [a --> c]'1;
 [b --> b,c]'2;
 [b --> c]'2;
}
```

Fig. 1. P-Lingua specification file for a P system with two membranes

the sequel the expressions of AP and L will not be explicitly given, the implication being that they are defined as above.

Additionally, a halt (sink) state is needed when H is not left-total and an extra atomic proposition, that indicates that the system has reached this state, is added to AP.

Definition 3. *An (infinite) path in a Kripke structure $M = (S, H, I, L)$ from a state $s \in S$ is an infinite sequence of states $\pi = s_0 s_1 \ldots$, such that $s_0 = s$ and $(s_i, s_{i+1}) \in H$ for every $i \geq 0$. A finite path π is a finite prefix of an infinite path.*

The set of all (infinite) paths from initial states is denoted by $Path(M)$. The set of all finite paths from initial states is denoted by $FPath(M)$.

2.4 Linear Temporal Logic (LTL)

The most widely used temporal specification languages in model checking are *Linear Temporal Logic* (LTL) [22, 23] and the branching time logic CTL (*Computation Tree Logic*) [5]. The superset of these logics is CTL* [9], which combines both linear-time and branching-time operators. A state formula in CTL* may be obtained from a path formula by prefixing it with a path quantifier, either an **A** (for every path) or an **E** (there exists a path).

In LTL the only path quantifier allowed is **A**, i.e. we can describe only one path properties per formula and the only state subformulas permitted are atomic propositions. More precisely, LTL formulas satisfy the following rules [6]:

- If $p \in AP$, then p is a path formula
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$ and $f\mathbf{R}g$ are path formulas, where:

- The **X** operator ("neXt time", also written ◯) requires that a property holds in the next state of the path.
- The **F** operator ("eventually" or "in the future", also written ◊) is used to assert that a property will hold at some state on the path.
- **G** ("always" or "globally", also written □) specifies that a property holds at every state on the path.
- The **U** operator ("until") holds if there is a state on the path where g holds, and at every preceding state on the path, f holds.
- **R** ("release") is the logical dual of the **U** operator. It requires that the second property holds along the path up to and including the first state where the first property holds.

2.5 NuSMV

NuSMV is a symbolic model checker [3], developed as part of a joint project between Carnegie Mellon University (CMU) and Istituto per la Ricerca Scientifica e Tecnologica (IRST). NuSMV is the result of the reengineering, reimplementation, and, to a limited extent, extension of the SMV model checker [18], developed by CMU. It is publicly available at http://nusmv.irst.itc.it/. NuSMV [3] can process files written in SMV (Symbolic Model Verifier) language [18] (the NuSMV language is mostly source compatible with the original version of SMV) and supports LTL and CTL as temporal specification logics.

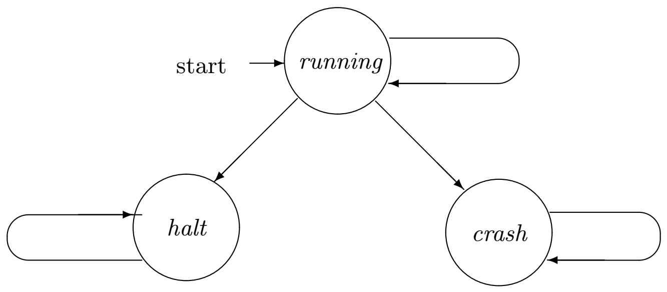


Fig. 2. Non-deterministic finite state machine

The input language of NuSMV was designed to allow descriptions of Finite State Machines (FSMs), more precisely to describe the transition relation of the FSM. This relation defines the valid evolutions of the FSM. For example, given the FSM from Fig. 2, the corresponding SMV code is:

```
MODULE main
VAR
  state : {running, halt, crash};
```

```
ASSIGN
  init(state) := running;
  next(state) := case
    state = running : {running, halt, crash};
    state = halt : halt;
    state = crash : crash;
  esac;
```

The transition relation of the FSM can be expressed also using the `TRANS` keyword. For more details refer to [3].

```
MODULE main
VAR
  state : {running, halt, crash};
ASSIGN
  init(state) := running;
TRANS
  state = running & next(state) = running |
  state = running & next(state) = halt |
  state = running & next(state) = crash |
  state = halt & next(state) = halt |
  state = crash & next(state) = crash
```

Having the model described in NuSMV, one can add LTL or CTL specifications to be verified by the model checker. For example, the specification `LTLSPEC G !( F state = halt)` is false and the counterexample obtained is the path: $running \to halt \to halt \to \ldots$ (the system will eventually remain in the *halt* state). On the other hand, the specification `LTLSPEC G !(state = halt & X state = running)` is true (there is no transition from the *halt* state, having the next state *running*).

3 Coverage criteria for P systems

A set of coverage criteria for P system rules, inspired from grammar testing, is presented in [12]. Test sets should be further designed to satisfy each coverage criterion. In the following we summarize the main coverage criteria, but for simplicity, we will provide the definitions only for one membrane P systems, $\Pi = (V, \mu, w, R)$, $\mu = [_1]_1$.

Definition 4. *A multiset denoted by $u \in V^*$,* covers *a rule $r : a \to v \in R$, if there is a computation $w \Longrightarrow^* xay \Longrightarrow x'vy' \Longrightarrow^* u$; $x, y, x', y', v, u \in V^*$, $a \in V$, $w \in V^*$ is the initial multiset. If there is no further computation from u, then this is called a* terminal coverage*.*

Definition 5. *A set $T \subseteq V^*$, is called a* test set *that satisfies the* rule coverage *(RC) criterion if for each rule $r \in R$ there is $u \in T$ which covers r. If every $u \in T$ provides a terminal coverage then T is called a test set that satisfies the* rule terminal coverage *(RTC) criterion.*

Definition 6. *A rule $r \in R$, $r : a \to ubv$, $u, v \in V^*$, $a, b \in V$, is called a* direct occurrence *of b. For every symbol $b \in V$, we denote by $Occs(\Pi, b)$, the set of all direct occurrences of b.*

Definition 7. *A multiset $z \in V^*$* covers *the rule $r : b \to y \in R$ for the* direct occurrence *of b, $a \to ubv \in R$, if there is a computation $w \Longrightarrow^* u_1 a v_1 \Longrightarrow u'_1 ubv v'_1 \Longrightarrow u''_1 u' y v' v''_1 \Longrightarrow^* z$; $u, v, u', v', u_1, v_1, u'_1, v'_1, u''_1, v''_1, y \in V^*$, $a, b \in V$. A set T_r is said to cover $r : b \to y$ for all direct occurrences of b if for any occurrence $o \in Occs(\Pi, b)$ there is $t \in T_r$ such that t covers r for o.*

Definition 8. *A set T is said to achieve* context-dependent rule coverage *(CDRC) for Π if it covers all $r \in R$ for all their direct occurrences. If every $z \in T$ provides a terminal coverage then T is called a test set that satisfies the* context-dependent rule terminal coverage *(CDRTC) criterion.*

To illustrate these concepts, we consider the P system $\Pi = (V, \mu, w, R)$ where: $V = \{a, b, c\}$, $\mu = [_1]_1$, $w = a$, $R = \{r_1 : a \to bc; r_2 : b \to bc; r_3 : b \to c\}$. It can be verified that the set $T = \{c^3\}$ covers all the rules, because the computation $a \Longrightarrow bc \Longrightarrow bc^2 \Longrightarrow c^3$ applies the rules r_1, r_2, r_3. Note that c^3 is a terminal configuration and, consequently, $T = \{c^3\}$ satisfies the RTC criterion. The test set $T' = \{bc^2, c^2\}$ achieves the CDRC criterion, because it covers the rules r_2, r_3, each one in the context defined by r_1. A test set satisfying the CDRTC is $T'' = \{c^2, c^3\}$.

4 Transforming a one-membrane P system into a Kripke structure

In a previous paper [15], a transformation of a one-membrane P system into a Kripke structure was proposed and several theoretical aspects were analysed. To simplify the presentation, we will consider one-membrane P system and show in next section how this approach can be extended to an arbitrary system.

Consider the one-membrane P system $\Pi = (V, \mu, w, R)$, where $R = \{r_1, \ldots, r_m\}$; each rule r_i, $1 \leq i \leq m$, is of the form $u_i \longrightarrow v_i$, where u_i and v_i are multisets over the alphabet V. In the sequel, we treat the multisets as vectors of non-negative integers, that is each multiset u is replaced by $\Psi_V(u) \in \mathbf{N}^k$, where k denotes the number of symbols in V; so, we will write $u \in \mathbf{N}^k$.

In order to define the Kripke structure associated to Π we use two predicates $MaxPar$ and $Apply$ (similar to [7]): $MaxPar(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$, $u \in \mathbf{N}^k$, $n_1, \ldots, n_m \in \mathbf{N}$ signifies that a computation from the configuration u in maximally parallel mode is obtained by applying rules $r_1 : u_1 \longrightarrow v_1, \ldots, r_m :$

$u_m \longrightarrow v_m$, $n_1, \ldots, n_m$ times, respectively (in particular, $MaxPar(u, u_1, v_1, 0, \ldots, u_m, v_m, 0)$ signifies that no rule can be applied and so u is a terminal configuration); $Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$, $u, v \in \mathbf{N}^k$, $n_1, \ldots, n_m \in \mathbf{N}$, denotes that v is obtained from u by applying rules $r_1, \ldots, r_m$, $n_1, \ldots, n_m$ times, respectively.

In order to keep the number of configurations finite, for each configuration $u = (u^{(1)}, ..., u^{(k)})$ we will assume that each component, $u^{(i)}$, $1 \leq i \leq k$ cannot exceed an established upper bound, denoted Max, and each rule can only be applied for at most a given number of times, denoted Sup.

We denote $u \leq Max$ if $u^{(i)} \leq Max$ for every $1 \leq i \leq k$ and $(n_1, \ldots, n_m) \leq Sup$ if $n_i \leq Sup$ for every $1 \leq i \leq m$; $\mathbf{N}^k_{Max} = \{u \in \mathbf{N}^k \mid u \leq Max\}$, $\mathbf{N}^m_{Sup} = \{(n_1, \ldots, n_m) \in \mathbf{N}^m \mid (n_1, \ldots, n_m) \leq Sup\}$. Analogously to [7], the system is assumed to crash whenever $u \leq Max$ or $(n_1, \ldots, n_m) \leq Sup$ does not hold (this is different from the normal termination, which occurs when $u \leq Max$, $(n_1, \ldots, n_m) \leq Sup$ and no rule can be applied). Under these conditions, the one-membrane P system Π can be described by a Kripke structure $M = (S, H, I, L)$ with $S = \mathbf{N}^k_{Max} \cup \{Halt, Crash\}$ with $Halt, Crash \notin \mathbf{N}^k_{Max}$, $Halt \neq Crash$; $I = w$ and H defined by:

- $(u, v) \in H$, $u, v \in \mathbf{N}^k_{Max}$, if $\exists (n_1, \ldots, n_m) \in \mathbf{N}^m_{Sup} \setminus \{(0, \ldots, 0)\} \cdot$ $MaxPar(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m) \wedge$ $Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$;
- $(u, Halt) \in H$, $u \in \mathbf{N}^k_{Max}$, if $MaxPar(u, u_1, v_1, 0, \ldots, u_m, v_m, 0)$;
- $(u, Crash) \in H$, $u \in \mathbf{N}^k_{Max}$, if $\exists (n_1, \ldots, n_m) \in \mathbf{N}^m, v \in \mathbf{N}^k \cdot$ $\neg((n_1, \ldots, n_m) \leq Sup \wedge v \leq Max) \wedge MaxPar(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m) \wedge$ $Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$;
- $(Halt, Halt) \in H$;
- $(Crash, Crash) \in H$.

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

The main result of [15] can be summarized by the following theorem:

Theorem 1. *Given an one-membrane P system $\Pi = (V, [_1]_1, w_1, R)$, (terminal) test suites satisfying the rule coverage and context dependent rule coverage criteria are generated based on LTL specifications.*

This means that having the transformation into a Kripke structure, what we need is to verify the following LTL specifications:

- $G\neg((n_i \geq 1) \wedge (state = other))$, for each rule $r_i \in R$, in order to achieve rule coverage (RC).
- $G\neg((n_i \geq 1) \wedge (state = other) \wedge F(state = halt))$, for each rule $r_i \in R$, in order to obtain rule terminal coverage (RTC).
- $G\neg((n_i \geq 1) \wedge X((n_j \geq 1) \wedge (state = other)))$, for each pair of rules $(r_i, r_j) \in R \times R$, where r_j can be applied the context of r_i, in order to achieve context dependent rule coverage (CDRC).

- $G\neg((n_i \geq 1) \wedge X((n_j \geq 1) \wedge (state = other)) \wedge F(state = halt))$, for each pair of rules $(r_i, r_j) \in R \times R$, where r_j can be applied the context of r_i, in order to obtain context dependent rule terminal coverage (CDRTC).

5 Generating the test suits

5.1 The test generation tool

Following the test generation strategy presented previously, a tool was developed, which functions as described by Fig. 3. A graphical interface allows editing and verification of PLI files, representing the specification of P systems in P-Lingua. The tool communicates with the P-Lingua framework, using its parser to syntactically verify the specification. Once the specification contains no syntactically errors, the corresponding objects from the library `pLinguaCore` are created. For a given `Psystem` object, an SMV file is created, containing the associated SMV model (corresponding to the Kripke structure). The user has the possibility to choose which of the following coverage criteria should be employed: RC, RTC, CDRC, CDRTC. For each coverage criteria LTL specifications, representing never-claim formulas, are added to the SMV file. Finally, the NuSMV model checker is run against the model specified in SMV, the counterexamples are decoded and transformed into test cases.

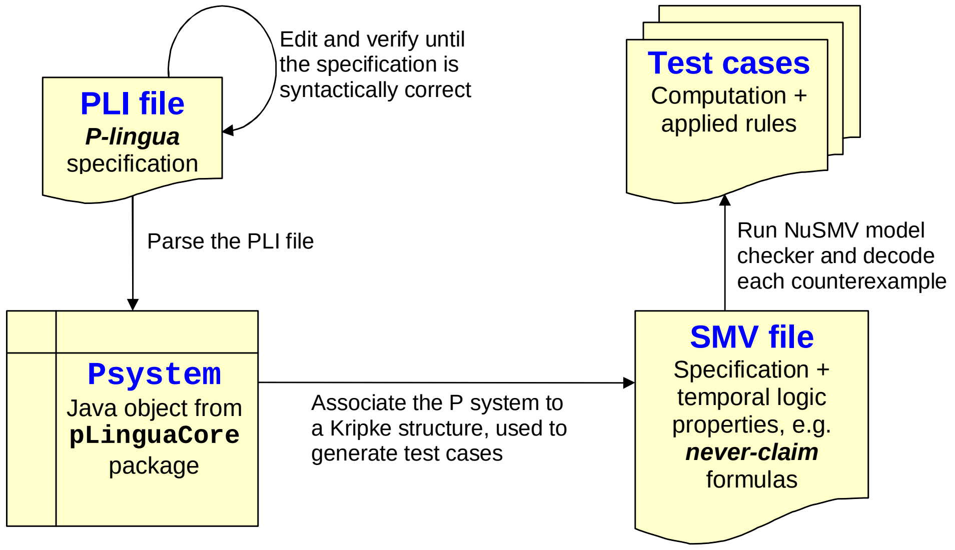

Edit and verify until the specification is syntactically correct
PLI file
P-lingua specification
Parse the PLI file
Psystem
Java object from pLinguaCore package
Associate the P system to a Kripke structure, used to generate test cases
SMV file
Specification + temporal logic properties, e.g. never-claim formulas
Run NuSMV model checker and decode each counterexample
Test cases
Computation + applied rules

Fig. 3. Tool overview

Having a P system specified in P-Lingua, the tool automatically transforms it into a SMV model. This model depends on the type of P system and in the following subsections we will present the transformation strategies proposed for different types of P systems. After the transformations the LTL specifications are automatically generated and appended to the SMV file.

5.2 Transforming one-membrane P systems into SMV

For an one-membrane P system $\Pi = (V, \mu, w, R)$, with $V = \{a_1, \dots, a_k\}$ and $R = \{r_1, \dots r_m\}$ (each rule r_i has the form $u_i \longrightarrow v_i$), its associated Kripke structure is $M = (S, H, I, L)$. The state space of M is implemented by using a 3-valued "state" variable (with values "Halt", "Crash" and "Running") and appropriate variables to hold the current configuration and the number of applications of each rule. Therefore, the NuSMV model will contain:

- k variables, labelled exactly like the objects from the alphabet V, each one showing the number of occurrences of each object, $a_i \in V$, $1 \leq i \leq k$;
- m variables n_i, $1 \leq i \leq m$, each one showing the number of applications of $r_i \in R$, $1 \leq i \leq m$;
- one variable *state* showing the current state of the model, $state \in \{Running, Halt, Crash\}$;
- two constants, Max corresponding to the upper bound for the number of occurrences expressed by each $a_i \in V, 1 \leq i \leq k$ and *Sup* which shows that each rule $r_i, 1 \leq i \leq m$, can be applied at most *Sup* times (see Section 4).

With these notations we are prepared to construct a NuSMV specification as a FSM where the states and transitions are defined below and also abstracted in Fig. 2.

If the current state is *Running* then this is characterised by the values provided by $a_1 \geq 0, \dots, a_k \geq 0$; the maximal parallelism condition will be written as a conjunction $c_1 \wedge \cdots \wedge c_m$, where each condition c_i, $1 \leq i \leq m$, corresponds to rule r_i and is a disjunction $c_i = c_{i_1} \vee \cdots \vee c_{i_p}$, given the left hand side of r_i is $a_{i_1}^{t_{i_1}} \dots a_{i_p}^{t_{i_p}}$. The condition c_{i_j}, $1 \leq j \leq p$, is $0 \leq a_{i_j} - n_1 h_1 - \cdots - n_m h_m < t_{i_j}$, where $n_1, \dots, n_m$ represent the values provided by $MaxPar$ and $h_q \geq 0$ represents the number of occurrences of symbol a_{i_j} on the left hand side of r_q. This condition simply states that, after applying all rules in a maximal parallel way, the number of occurrences of symbol a_{i_j} left is less than the number of occurrences of a_{i_j} appearing on the left hand side of r_i, i.e., this rule can no longer be applied for this step. When the number of occurrences of the symbol a_{i_j} in the left side of a rule r_q is equal to 1, then the above inequality $0 \leq a_{i_j} - n_1 h_1 - \cdots - n_m h_m < t_{i_j}$ becomes $0 = a_{i_j} - n_1 h_1 - \cdots - n_m h_m$ (because $t_{i_j} = 1$).

The values $a_1 \geq 0, \dots, a_k \geq 0$ that characterise the next state are computed as follows. Using the above notations and denoting by $next(a)$ the new value, we have $next(a_{i_j}) = a_{i_j} - n_1 h_1 - \cdots - n_m h_m + n_1 h'_1 + \cdots + n_m h'_m$, where $h'_q \geq 0$ represents the number of occurrences of symbol a_{i_j} on the right hand side of r_q.

Some additional conditions are added to the above ones in order to distinguish the destination state. These are obvious and derive from the upper bound conditions introduced. The example below illustrates the approach. We notice that all these conditions and the entire NuSMV specification, including the LTL expressions, are automatically derived from a P system using the tool developed by the authors of this paper.

We illustrate the approach by using the following one-membrane P systems: $\Pi_1 = (V_1, \mu, w_1, R_1)$, having $V_1 = \{s, a, b, c\}$, $\mu = [_1]_1$, $w_1 = s$, $R_1 = \{r_1 : s \rightarrow ab; r_2 : a \rightarrow c; r_3 : b \rightarrow bc; r_4 : b \rightarrow c\}$ and $\Pi_2 = (V_2, \mu, w_2, R_2)$, having $V_2 = \{s, a, b, c, d, x\}$, $\mu = [_1]_1$, $w_2 = s$, $R_2 = \{r_1 : s \rightarrow abc; r_2 : ab \rightarrow d^2; r_3 : c \rightarrow ab; r_4 : abd^2 \rightarrow x\}$.

The transition from the state *Running* to itself, for the P system Π_1, which has non-cooperative rules, can be written as the following NuSMV specification, where the second row shows that all the objects have been consumed and no rule can be further applied (maximal parallelism):

```
state = running & next(state) = running &
s - next(n1) = 0 & a - next(n2) = 0 & b - next(n3) - next(n4) = 0 &
next(s) = s - next(n1) &
next(a) = a - next(n2) + next(n1) &
next(b) = b - next(n3) - next(n4) + next(n1) + next(n3) &
next(c) = c + next(n2) + next(n3) + next(n4) &
! (next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0) &
! (next(s) > Max | next(a) > Max | next(b) > Max | next(c) > Max |
next(n1) > Sup | next(n2) > Sup | next(n3) > Sup | next(n4) > Sup)
```

The maximal parallelism condition for Π_2, a P system with *cooperative rules*, becomes a conjunction of disjunctions $c_1 \wedge \cdots \wedge c_m$, each c_i corresponding to a rule:

```
(s-next(n1)=0) & (a-next(n2)-next(n4)=0 | b-next(n2)-next(n4)=0) &
(c-next(n3)=0) & (a-next(n2)-next(n4)=0 | b-next(n2)-next(n4)=0 |
(0<=d-2*next(n4) & d-2*next(n4)<2))
```

When one specification is false, a counterexample is given, i.e. a trace of the FSM that falsifies the property. Based on the counterexample received for the specification `G !((n1 > 0 &  X n2 > 0) & F state = halt)` of Π_1, a test sequence checking that r_2 appears in the context of r_1 on a terminal computation starting with w is obtained. This is given by $s \Longrightarrow ab \Longrightarrow c^2$ and the rules applied are r_1 first and r_2, r_4 at the second step.

In the following we will present an excerpt of a counterexample received from NuSMV, for the P system Π_1, edited for brevity:

```
-- specification G !((n3 > 0 & X n3 > 0) & F state = halt) is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
```

```
-> State: 8.1 <-          s = 0                 c = 3
  s = 1                   a = 1                 n2 = 0
  a = 0                   b = 1               -> State: 8.5 <-
  b = 0                   n1 = 1                b = 0
  c = 0                 -> State: 8.3 <-        c = 4
  n1 = 0                  a = 0                 n3 = 0
  n2 = 0                  c = 2                 n4 = 1
  n3 = 0                  n1 = 0              -- Loop starts here
  n4 = 0                  n2 = 1              -> State: 8.6 <-
  state = running         n3 = 1                n4 = 0
-> State: 8.2 <-        -> State: 8.4 <-        state = halt
```

The values of all variables are listed only once, for the first configuration of the counterexample. Then, at the following steps, only the modified variables are printed. Based on the counterexample received for the specification `G !((n3 > 0 &  X n3 > 0) & F state = halt)`, the tool computes the entire configuration at each step and the applied rules. The test case corresponding to the use of rule r_3 in the context of r_3, is represented by the P system derivation: $s \Longrightarrow ab \Longrightarrow bc^2 \Longrightarrow bc^3 \Longrightarrow c^4$. The rules used were: first r_1, then r_2, r_3, for the third transition r_3 and finally r_4, as it can be seen from the following table, corresponding to the counterexample above:

State	s	a	b	c	n1	n2	n3	n4	state
8.1	1	0	0	0	0	0	0	0	running
8.2	0	1	1	0	1	0	0	0	running
8.3	0	0	1	2	0	1	1	0	running
8.4	0	0	1	3	0	0	1	0	running
8.5	0	0	0	4	0	0	0	1	running
8.6	0	0	0	4	0	0	0	0	halt

5.3 Transforming multi-membrane P systems into SMV

The transformation of multi-membrane P systems into SMV is similar to the one for one-membrane P systems. The differences are the following:

- If the P system contains $p > 1$ membranes, the SMV model will contain $k \times p$ variables for the occurrences of the objects in each membrane; labelled like the symbols from the alphabet V, $|V| = k$, with an additional index, representing the membrane.
- The variables n_i, $1 \leq i \leq |R_1| + \ldots + |R_m|$ will be used to represent the number of applications of each rule r_i, (we have considered that the rules from all membranes are labelled $r_1, \ldots, r_{|R_1|+\ldots+|R_m|}$).

We will illustrate these differences, compared to the one-membrane P system, by specifying in NuSMV the P system $\Pi_3 = (V, \mu, w_1, w_2, R_1, R_2)$, $V = \{s, a, b, c\}$, $\mu = [_1[_2]_2]_1$, $w_1 = s$, $w_2 = \lambda$, $R_1 = \{r_1 : s \rightarrow sa(b, in); r_2 : s \rightarrow ab; r_3 : b \rightarrow a; r_4 :$

$a \rightarrow c\}$, $R_2 = \{r_5 : b \rightarrow bc, r_6 : b \rightarrow c\}$, whose P-Lingua specification is given in Fig. 1.

The SMV model has the variables: $s_1, a_1, b_1, c_1, s_2, a_2, b_2, c_2, state$, the constants Max, Sup and the differences from the previous model will be given by the communications between membranes. For example, the number of objects b in the inner membrane, labelled with b_2, will take into account: the number of b objects consumed in membrane 2 by applying rules r_5, r_6, the number of objects b produces by the rules r_5 in membrane 2 and r_1 in membrane 1. An excerpt, representing the self loop from the state *running* is the following (edited for brevity):

```
state = running & next(state) = running &
s1 - next(n1) - next(n2) = 0 & b1 - next(n3) = 0 & a1 - next(n4) = 0 &
b2 - next(n5) - next(n6) = 0 &
next(s1) = s1 - next(n1) - next(n2) + next(n1) &
next(a1) = a1 - next(n4) + next(n1) + next(n2) + next(n3) &
next(b1) = b1 - next(n3) + next(n2) &
next(c1) = c1 + next(n4) &
next(s2) = s2 &
next(a2) = a2 &
next(b2) = b2 - next(n5) - next(n6) + next(n1) + next(n5)&
next(c2) = c2 + next(n5) + next(n6) &
! (next(n1) = 0 & ... & next(n6) = 0 ) &
! (next(s1) > Max | ... | next(c1) > Max |
   next(s2) > Max | ... | next(c2) > Max |
   next(n1) > Sup | ... | next(n6) > Sup )
```

5.4 Transforming P systems with dissolving rules into SMV

Similarly to the previous section, this P system model, which has $p > 1$ membranes, will have $k \times p$ variables to represent the occurrences of the objects in each membrane, a number of variables n_i equal to the total number of rules and some special variables, to mark the membranes affected by dissolving rules.

For each membrane that can be dissolved we will consider a variable $dis_i \in \{0, 1\}$, showing whether the membrane is dissolved (1) or it is still alive (0). When the membrane is dissolved its objects are assimilated by the outer membrane. In the SMV transformation we have used the variables dis_i as flags, to correctly update the values of the variables counting the occurrences of objects in each membrane.

Consider the P system $\Pi_4 = (V, \mu, w_1, w_2, R_1, R_2)$, $V = \{a, b, c\}$, $\mu = [_1[_2]_2]_1$, $w_1 = b$, $w_2 = abc$, $R_1 = \{r_1 : b \rightarrow c; r_2 : c \rightarrow a\}$, $R_2 = \{r_3 : b \rightarrow ab; r_4 : b \rightarrow b\delta, r_5 : c \rightarrow cc\}$. The inner membrane, labelled 2, can be dissolved when the rule r_4 is applied. In the SMV model the variable dis_2 is updated by the instruction:

```
next(dis2) := case
    dis2 = 1 : 1;      -- if membrane is dissolved, it remains dissolved
    next(n4) >= 1 : 1; -- if rule r4 is applied, membrane 2 dissolves
    1 : 0;             -- otherwise, the membrane remains alive (0)
```

```
esac;
```

If the membrane is dissolved (`dis2=1`), then it remains dissolved. When rule r_4 is applied (`next(n4)>=1`) the membrane will dissolve (next value is 1). Otherwise (`1` means *true* in this case and shows the default option), the membrane will remain alive, the next value is 0 (not dissolved).

An excerpt from the SMV file, showing the transition from the *running* state to itself is given below. To ensure that, only when a membrane is dissolved its content is moved to the outer membrane, the rules for updating the quantities in the outer membrane, e.g. `next(a1)`, have added an extra term, e.g. `a2*next(dis2)`, representing the same type of objects from the inner membrane, multiplied with the next value `dis2`, because this term is null when the membrane is alive and it is exactly `a2` when the membrane dissolves. On the other hand, the values from the inner membrane will be multiplied with the opposed value, `(1-next(dis2)`. When membrane 2 dissolves, the number of objects in membrane 2 will become 0 because the factor `(1-next(dis2)` is 0. Otherwise, the factor is 1 and the value is computed as usual (subtracting the consumed objects and adding the produced ones).

```
state = running & next(state) = running &
b1 - next(n1) = 0 & c1 - next(n2) = 0 &
b2 - next(n3) - next(n4) = 0  & c2 - next(n5) = 0 &
next(a1) = a1 + next(n2) + a2*next(dis2) &
next(b1) = b1 - next(n1) + b2*next(dis2) &
next(c1) = c1 - next(n2) + next(n1) + c2*next(dis2) &
next(a2) = (a2 + next(n3))*(1-next(dis2)) &
next(b2) = (b2-next(n3)-next(n4)+next(n3)+next(n4))*(1-next(dis2)) &
next(c2) = (c2 - next(n5) + 2*next(n5))*(1-next(dis2)) &
! (next(n1) = 0 & ... & next(n5) = 0)&
! (next(a1) > Max | ... | next(c2) > Max |
   next(n1) > Sup | ... | next(n5) > Sup )|
```

5.5 Using the transformed model to verify different properties

The transformation of a P systems into a model accepted by a specific model-checker, NuSMV, was used to generate test cases. For this, LTL specifications were written, such as `G !(n_2 > 0 &  F(state = halt))`, having the purpose of obtaining a terminal coverage for rule r_2. If the LTL specification is false, the counterexample obtained is decoded: it represents a path in the SMV model, which corresponds to a (possible partial) computation in the P system. The union of all test cases will form the test suite.

If an LTL specification like `G !(n2 > 0 & F(state = halt))` is true this means that: (1) rule r_2 is never applied or (2) rule r_2 is applied, but the computation does not finish, i.e. the system does not reach the *halt* state. Verifying the simpler specification `G  !(n2 > 0)` and receiving from the model checker the response 'specification is true' reveals the fact that this rule is never applied. This is normally a fault in the model and could be obtained in situations like the following:

`@mu = [ ]'1;`	`@mu = [ ]'1;`
`@ms(1) = s;`	`@ms(1) = s;`
`[s --> a,b]'1;`	`[s --> a,b ]'1;`
`[a --> c]'1;`	`[x --> c]'1;`
`[b --> b,c]'1;`	`[b --> b,c]'1;`
`[b --> c]'1;`	`[b --> c]'1;`

In the left column is given a correct specification in P-Lingua of a certain P system; on the right side the second rule has a typo (x instead of a). Both fragments are syntactically correct, so the parser will accept them both. On the other hand the second P system has different computations and rule r_2 is never applied, fact revealed by the specification `G !(n2 > 0)`.

The automatic transformation to NuSMV allows verifications of different LTL or CTL specifications, that might be useful at designing a P system, that models a certain process. Even simple propositions like `G !(a > 100)` let us know if there exist or not a computation in which the number of objects a can reach a certain level. And this answer is obtained quickly, without simulating hundreds of computations to see if this ever happens.

5.6 Test generation using bounded model checking

Model checking tools face a combinatorial blow up of the state-space, known as the *state explosion* problem. This can occur if the system being verified has many components which can make transitions in parallel [5]. As P systems work in parallel and have a non-deterministic nature, the number of global system states may grow exponentially. One approach to alleviate this problem is based on using *Binary Decision Diagrams* (BDD) [18], this being the case of NuSMV, which implements BDD model checking.

Another approach to face the state explosion problem is to use *Bounded Model Checking* (BMC) algorithms. NuSMV provides also a BMC mode: it tries to find a counterexample of increasing length, and stops when it succeeds, declaring that the formula is false. The maximum number of iterations can be specified, the default value is 10. For testing it is preferable to obtain shorter test cases, so the BMC option can be very useful at test generation.

It should be emphasized that: if the maximum number of iterations is reached and no counterexample is found, the truth of the formula is not decided. In this case we cannot conclude that the formula is true, but only that any counter-example should be longer than the maximum length.

As a final conclusion, for test generation the BMC option of NuSMV is very useful. For verifying other properties of the P system NuSMV should be used in the default mode.

6 Related work

Only a few approaches on model checking P systems have been proposed until now. Among them, decidability of model checking problems for P systems was analysed and a discussion regarding the use of SPIN model checker provided [7], for P systems without priority rules and membrane dissolving rules. An operational semantics using rewriting logics and model checking based on Maude was given in [1]. Regarding probabilistic model checking of P systems Romero-Campero et al. proposed in [24] a transformation into a probabilistic and symbolic model checker called PRISM.

Several testing strategies for P systems have been presented, that use: coverage criteria inspired from grammar testing [12], finite state based testing [13] and stream X-machine models [14]. An approach to automate the test generation for P systems using model checking is presented in [15] and further extended in this work.

7 Conclusions

This paper extends our previous work on model-checking based P system testing [15]. It integrates this approach with P-Lingua, a software framework for cell-like P systems [11]. Compared to the previous work, the current tool offers support for multi-membrane P systems. Also, we propose an approach for transforming P systems with dissolving rules into NuSMV. The transformed model can be used not only for test generation, but also for verifying system properties. A reduction in test case size can be obtained if bounded model checking is used.

Future work will focus on other types of P systems, employed in modelling ecosystems [2], for which automatic test generation and model checking would be very useful. We will study testing and verification of probabilistic and stochastic P systems, integration with P-Lingua and possible use of a probabilistic model checker, e.g. PRISM, as suggested in [24].

Another research topic concerns the study of other model checkers and improvements made to the strategies presented, to face the *state explosion* problem, which appears when more complex models are verified.

Acknowledgements

This work was supported by CNCSIS - UEFISCSU, project number PNII IDEI 643/2008.

References

1. O. Andrei, G. Ciobanu, and D. Lucanu. A rewriting logic framework for operational semantics of membrane systems. *Theoretical Computer Science*, 373(3):163–181, 2007.
2. M. Cardona, M. A. Colomer, A. Margalida, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and D. Sanuy. A P system based model of an ecosystem of some scavenger birds. In Păun et al., editors. *Membrane Computing, 10th International Workshop, WMC 2009, Revised Selected and Invited Papers*, volume 5957 of *Lecture Notes in Computer Science*. Springer, pages 182–195, 2010.
3. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.
4. G. Ciobanu, M, J. Pérez-Jiménez, and G, Păun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.
5. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer, pages 52–71, 1981.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
7. Z. Dang, O. H. Ibarra, C. Li, and G. Xie. On the decidability of model-checking for P systems. *Journal of Automata, Languages and Combinatorics*, 11(3):279–298, 2006.
8. D. Díaz-Pernil, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A P-Lingua programming environment for membrane computing. In Corne et al., editors, *Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers*, volume 5391 of *Lecture Notes in Computer Science*. Springer, pages 187–203, 2009.
9. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *STOC '82: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. ACM, pages 169–180, 1982.
10. G. Fraser, F. Wotawa, and P. Ammann. Testing with model checkers: a survey. *Software Testing, Verification and Reliability*, 19(3):215–261, 2009.
11. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. An overview of P-Lingua 2.0. In Păun et al. , editors. *Membrane Computing, 10th International Workshop, WMC 2009. Revised Selected and Invited Papers*, volume 5957 of *Lecture Notes in Computer Science*. Springer, pages 264–288, 2010.
12. M. Gheorghe and F. Ipate. On testing P systems. In Corne et al., editors, *Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers*, volume 5391 of *Lecture Notes in Computer Science*. Springer, pages 204–216, 2009.
13. F. Ipate and M. Gheorghe. Finite state based testing of P systems. *Natural Computing*, 8(4):833–846, 2009.
14. F. Ipate and M. Gheorghe. Testing non-deterministic stream X-machine models and P systems. *Electronic Notes in Theoretical Computer Science*, 227:113–126, 2009.
15. F. Ipate, M. Gheorghe, and R. Lefticaru. Test generation from P systems using model checking. *Journal of Logic and Algebraic Programming*, DOI:10.1016/j.jlap.2010.03.007, in press, 2010.

16. R. Lefticaru, M. Gheorghe, and F. Ipate. An empirical evaluation of P system testing techniques. *Natural Computing*, DOI:10.1007/s11047-010-9188-y, in press, 2010.
17. M. A. Martínez-del-Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A P-Lingua based simulator for tissue P systems. *Journal of Logic and Algebraic Programming*, DOI:10.1016/j.jlap.2010.03.009, in press, 2010.
18. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publ., 1993.
19. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. Turku Center for Computer Science-TUCS Report 208, November 1998.
20. G. Păun. *Membrane Computing: An Introduction.* Springer-Verlag, 2002.
21. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
22. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.
23. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
24. F. J. Romero-Campero, M. Gheorghe, L. Bianco, D. Pescini, M. J. Pérez-Jiménez, and R. Ceterchi. Towards probabilistic model checking on P systems using PRISM. In Hoogeboom et al., editors, *Membrane Computing, 7th International Workshop, WMC 2006, Revised, Selected, and Invited Papers*, volume 4361 of *Lecture Notes in Computer Science*, pages 477–495. Springer, 2006.

Modeling and Analysis of Firewalls by (Tissue-like) P Systems

Alberto Leporati, Claudio Ferretti

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,ferretti}@disco.unimib.it

Summary. We propose to use tissue-like P systems as a tool to model and analyse the security properties of firewall systems. The idea comes from a clear analogy between firewall rules and P systems rules: they both modify and or move objects (data packets, or symbols of an alphabet) among the regions of the system. The use of P systems for modeling packet filters, routers and firewalls gives the possibility to check — and possibly mathematically prove — some security properties.

1 Introduction

Firewalls are devices that allow to control network traffic. Depending on the protocol layer they operate at, firewalls can be classified into packet filters, circuit proxies, and application level proxies. Since they allow to enforce security policies, firewalls are essential for organizations that are connected to the Internet, and/or whose networks are divided in a number of segments. In fact, they operate like filters that selectively choose what data packets are allowed to cross the boundaries between network segments, and thus ensure that information flow between those segments only in the intended ways.

When deploying firewalls in an organization, it is essential to verify that they are configured properly. Unfortunately, firewall configurations are often written in a low-level language which is hard to understand. Thus, it is often quite difficult to find out which connections and services are actually allowed by the configuration. Indeed, it is well recognized that writing a correct set of rules is a challenging task. Hence, network administrators would benefit greatly using a tool that helps them to analyze the behavior of firewall rules.

Membrane systems (also known as *P systems*) are a distributed, parallel and synchronous model of computation inspired by the functioning of living cells [15]. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. At least two ways to apply the rules are considered in the literature: the *maximally parallel* and the *sequential* way. When two or more (sets of) rules can be applied in a given computation step, a nondeterministic choice is performed. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be executed. *Tissue* P systems [10, 11] can be viewed as an evolution of P systems, corresponding to a shift from cell-like to tissue-like architectures, based on intercellular communication and cooperation between cells. In this model cells are usually composed of a single membrane, and the interconnection structure forms an arbitrary graph. The cells are the nodes of the graph, and objects may either evolve by means of *evolution rules*, or move between cells (alongside the edges of the graph) as a result of the application of *symport/antiport* or *uniport* rules. In what follows we assume the reader is familiar with the basic notions and the terminology underlying P systems. For details, and a systematic introduction on the subject, we refer the reader to [16, 17]. The latest information about P systems can be found in [14].

By looking at firewall rules as filters that selectively choose what data packets are allowed to cross the boundary between two regions of the network, it is apparent that a firewall operates like a semi-permeable membrane that separates the two regions. Hence, membrane systems are easily seen as a natural tool that allow to model and analyze firewall systems.

In this paper we apply the membrane computing paradigm to the problem of properly configuring a collection of firewall systems. The scenario we imagine is the following: a network administrator has to devise the rules that allow to control traffic in a large network, composed of several segments. Each segment delimits an area, or zone, that contains hardware equipments such as PCs, servers, printers, etc. Each pair of adjacent areas is separated by a firewall, that for each direction selectively filters what data packets are allowed to cross the boundary in that direction. We assume that firewalls operate as packet filters, which are allowed to examine and possibly modify the fields of IP packets. This means, in particular, that they are also able to operate as *routers*: for example, they can modify the destination address of all packets having a specified source address and destination port. The aim is to help network administrators in testing and analyzing firewall rules before implementing them; in fact, usually the implementation must be performed quickly, since as soon as all network equipments are mounted the organization wants to start using the network. Moreover, it is always desirable to test a configuration change before putting it at work in a real network.

Another goal of our work — not addressed in this paper — is to give the possibility to *mathematically prove* security properties (such as, for example, the impossibility for a certain kind of TCP packets to reach a given region of the network). This will be obtained by considering *reachability* problems on the P systems that simulate the functioning of the firewalls. So doing, we will be able to find answers to questions like:

- What is the action for a specified IP packet?

- What packets are permitted by this list of rules?
- From which sources are packets to this destination permitted?
- Which services are accessible on a given host?
- Is a given host/network accessible from another given host/network?
- What kind of traffic is allowed between two networks?
- From which networks is a given host accessible?

Of course, some attempts have already been made in the literature to provide tools that help network administrators to test and analyze firewalls and packet filters before implementing them. To the best of our knowledge, no one of these attempts uses membrane systems. Moreover, much of the work currently present in the literature focuses on the configuration of a single firewall, and proposes algorithms that detect common configuration mistakes, such as rule shadowing, correlation, generalization and redundancy. The work whose spirit is most similar to ours has been done by Eronen and Zitting [1]; in their paper they describe an expert system whose knowledge base contains a representation of a firewall configuration. The tool allows to model a single firewall, but it has the advantage that it can also compare the firewall configuration against a list of known vulnerabilities and attacks, and warn the user whenever a match is found. Mayer, Wool and Ziskind [12] propose a firewall analysis engine based on graph algorithms. Similar work based on a logic background has been done by Hazelhurst et al. [4, 5, 6], where ordered binary decision diagrams have been used to analyze routers' access control lists. This representation allows for efficient handling of the lists: for instance, finding redundant (shadowed) rules is easy. Several researches have also implemented tools for describing and generating the contents of an access list. For example, Guttman [2] describes an approach for generating filters' rules starting from a desired security policy, and verifying that a packet filter correctly implements a given security policy.

The paper is organized as follows. In section 2 we briefly describe the features of packet filters we want to model, and consequently we derive some properties and constraints of the P systems we will use. In sections 3 and 4 we present the model of tissue-like P systems that results from this analysis, and we show how this model can be simulated by a single-membrane P system. Section 5 contains the conclusions, and gives some directions for future research.

2 Preliminary Analysis

We focus our attention to *stateless* firewalls with packet inspection, also known as *packet filters*. We assume the reader has some background in IP-based networking [19].

As told in the Introduction, firewalls function as routers which connect different network segments together. Based on their configuration, they may restrict the traffic flowing between the different segments. Despite being so common, firewalls, routers, and many other simple packet filters usually lack good user interfaces

```
1 permit udp any host 192.168.1.1 eq 53
2 deny udp any host 192.168.1.2
3 permit udp any 192.168.1.0 0.0.0.255 eq 123
4 permit udp any host 192.168.1.2 eq 177
5 deny ip any any
```

Fig. 1. An example of a Cisco router access list. Note that the fourth rule is never matched because of the second rule

for specifying the desired security policy. Hence it is very easy to make mistakes when writing the lists of filtering rules, especially when these lists are long (several hundreds rules is not uncommon). In particular, it is possible to make four kinds of errors when implementing a security policy as a set of rules: *shadowing*, *correlation*, *generalization*, and *redundancy*. Briefly, a rule r_1 *shadows* a rule r_2 when r_1 is always executed before r_2, and r_2 operates on a subset of the IP packets which are processed by r_1. In this situation r_2 can never be executed and hence is useless. A pair of rules r_1 and r_2 are *correlated* when the sets of packets processed by them are not one the subset of the other but have a nonempty intersection; the problem arises in particular when the actions (either `accept` or `reject`) specified in the two rules are different. A rule r_2 is a *generalization* of a rule r_1 if r_2 follows r_1 in the order, and r_2 matches a superset of the packets matched by r_1, and the actions of r_2 and r_1 are different. Generalization may not be a configuration error, since it is generally accepted that more specific rules are followed by more general rules; however, since the actions performed are different it is a good practice to warn the administrator, so that he is aware of the situation. A *redundant* rule performs the same action on the same packets as another rule such that if the redundant rule is removed, the security policy will not be affected. Rule r_2 is redundant to rule r_1 if r_1 precedes r_2 in the order, and r_2 matches a subset (or the same set) of packets matched by r_1, and the actions of r_1 and r_2 are the same. If r_1 precedes r_2 and r_1 matches a subset of the packets matched by r_2, and the actions of r_1 and r_2 are the same, then rule r_1 is redundant to rule r_2 provided that r_1 is not involved in any generalization or correlation anomalies with other rules preceding r_2.

Since the administrator may write several rules that may affect the same set of IP packets, when one of these packets arrives the firewall must choose what rule to apply. Since a nondeterministic choice is not desirable, the firewall disambiguates by considering the rules in the same order as they have been specified, and applies the first rule whose parameters match the packet. For instance, Figure 1 shows an example of a Cisco router access list. When a packet is received, the list is scanned from the start to the end, and the action (either `permit` or `deny`) associated with the first match is taken. If a packet does not match any of the rules, the default action is `deny`. Often a "deny all" rule is included at the end of the list to make it easier to verify that a list has not been truncated. Separate lists can be specified for each network interface. As told above, it is very easy to make mistakes when

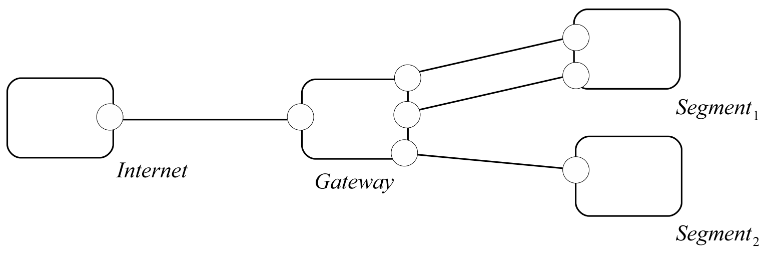


Fig. 2. The structure of a firewall P system. Each node of the graph represents a membrane in our tissue-like P system, that on its own represents a segment or zone of the network. The edges show the connections between zones. The small circles located on the membranes represent the lists of rules that filter the packets incoming to the membranes through the corresponding edges

writing access lists. For instance, the fourth rule in Figure 1 is never matched because it is shadowed by the second rule.

A crucial observation is that IP packets are processed one at the time, as soon as they arrive. Stated otherwise, network traffic is seen as a *stream* of packets. No cooperation among packets and no form of parallelism exist in current firewalls. If we sum all these observations, we can see that our model of P systems should apply *non-cooperative* rules in the *sequential* way; moreover, we should either impose that they are deterministic, or otherwise we should associate a linear (i.e., total) ordering $<$ to rules, so that if two rules $r_1 < r_2$ can be applied to a given object then r_2 (the rule which has the hightest priority) will be applied. However the assumption that our P systems are deterministic is not realistic, since this would mean that the above mentioned problems in writing firewall rules (shadowing, correlation, generalization, redundancy) have been solved *before* modeling the firewall by using the P system. Since we would rather like to use our P systems to solve these problems (as well as to answer questions concerning other security properties) we will assume that each rule has an associated priority.

Our rules will be an abstraction of the `accept` (also `permit`) and `reject` (also `deny`) rules which can be found in many packet filters such as, for example, Cisco routers [7], IPCHAINS and IPTABLES [13]. The rules will use the following fields from the IP protocol header: next level protocol (e.g., TCP, UDP or ICMP), source and destination IP addresses. In addition, some fields for upper level protocols, such as TCP and UDP port numbers, can be used. It will also be possible to specify entire subnets in place of single IP addresses, and to use wildcards when specifying the protocol or port numbers.

Since a firewall checks packets that try to pass the barrier in both directions, we should provide a separate list for the packets that try to enter and those trying to leave each of the regions separated by the firewall.

Another aspect that we have to take into account is that the segments of a network may be interconnected in an arbitrary way. Further, two segments may possibly be connected by two (or even more) network interfaces, that is, two segments may have multiple connections. This means that if we want to associate a membrane to each segment of the network we have to use a model similar to tissue P systems. In what follows we refer to this model — that will be formally defined in the next section — as *firewall P systems.* Figure 2 represents a sketch of the structure of a firewall P system. Nodes are labelled with the name of the segment or zone of the network, and edges show the existing connections between zones. The small circles located on the borders of the membranes represent the lists of rules that filter the packets incoming to the membranes through the corresponding edges. So, for example, $Segment_1$ is a zone connected to *Gateway* by means of two network interfaces. The two small circles located on the membrane of $Segment_1$ represent the two lists of rules that control the *incoming* traffic in the zone named $Segment_1$. The corresponding small circles located on the membrane named *Gateway* represent the lists of rules that filter the packets that come from $Segment_1$ and try to enter the zone *Gateway.* The precise form of the rules that compose these lists, as well as of the objects upon which these rules operate, is the subject of the next section.

3 The P Systems Model

In this section we give a precise definition of all the ingredients of our model of firewall P systems.

The *objects* represent IP packets. For the purposes of this paper, IP packets are represented as six-tuples

(*protocol, src_IP, dst_IP, src_port, dst_port, gateway*), where:

- *protocol* ∈ {TCP, UDP, ICMP};
- *src_IP* and *dst_IP* represent the source and destination address in the usual form (a quartet of integer numbers, each in the range 0..255);
- *src_port* and *dst_port* are integer numbers in the range 0..65535 that represent the source and the destination port associated with the source and destination address, respectively. These fields are meaningful only when *protocol* ∈ {TCP, UDP};
- *gateway* is an identifier of the edge to be followed at the next hop. It is used for routing purposes.

Note that in real IP packets there are other fields (such as *flags*) that we do not consider in this paper.

A *rule* is a quadruple of the form

(*priority, action, in_fields, out_fields*), where:

- *priority* is a non-negative integer that specifies the priority of the rule;
- *action* ∈ {`accept, reject, drop`} specifies how the packet filter should treat packets matched by the rule;

- *in_fields* is a six-tuple of the form

  (*protocol, src_IP/subnet, dst_IP/subnet, src_port, dst_port, gateway*),

  where:
  - *protocol* $\in$ {TCP, UDP, ICMP, `any`}. The value `any` is treated as a wildcard;
  - *src_IP* and *dst_IP* represent the source and destination address in the usual form (a quartet of integer numbers, each in the range 0..255). The subfield *subnet* is an integer in the range 0..32, and allows to specify subnets (that is, sets of IP addresses) in the customary way: it indicates that the specified number of bits of the IP address — starting from the most significant bit — are fixed (as specified in the address), whereas the others may assume any value. So, for example, the subnet 192.168.1.0/30 is composed by the IP addresses 192.168.1.x, where $x \in \{0, 1, 2, 3\}$.
    Alternatively, *src_IP* and/or *dst_IP* may assume the special value (wildcard) `any`. In such a case, the subfield *subnet* is not specified;
  - *src_port* and *dst_port* are integer numbers in the range 0..65535 that represent the source and the destination port (if *protocol* $\in$ {TCP, UDP}) associated with the source and destination address/subnet, respectively. Also in this case, *src_port* and/or *dst_port* may assume as a value the wildcard `any`;
  - *gateway* is an identifier (that is, a label) of the edge to which the rule is associated;
- *out_fields* is a six-tuple of the form

  (*protocol, src_IP, dst_IP, src_port, dst_port, gateway*),

  whose fields and values are defined exactly as those appearing in the objects of the system. Additionally, each field may assume the special value `same`, which indicates that the rule does not change the value of the field (that is, the value already present in the analyzed object is kept).

A rule r *matches* an object o if the fields specified in *in_fields* match. The fields *protocol*, *src_port* and *dst_port* match if they are equal (in the object and in the rule), or if the field in the rule contains the wildcard `any`. The field *src_IP* of o matches the field *src_IP/subnet* of r if the former IP address is contained in the subnet (set of IP addresses) of the latter. This definition includes the case in which the rule specifies a single IP address. The same criteria are applied when matching the field *dst_IP* of o with the field *src_IP/subnet* of r. The fields *gateway* match if and only if they are equal (in the object and in the rule).

The lists of rules are associated to the membranes, rather than to the regions enclosed by them, as is customary in membrane computing. Each list of rules checks the packets coming from a specific region, seeking to enter into the region enclosed by the membrane that contains the rules. As stated above, each list of rules is processed in the order given by priorities, until a match is found. The first matching rule specifies the action taken by the filter on the given object. In case two or more rules having the same priority match (a situation that should not occur, since it denotes a misconfiguration), a nondeterministic choice is made between such rules.

The application of an `accept` rule to an object o has the effect of letting the object enter the zone delimited by the membrane that contains the rule. Precisely, the object is removed from the system and a new object is created, with the same values of the fields as those of o but for the fields of *out_fields* different from `same`, which are rewritten with the new values specified in the rule. So doing it is possible to simulate *port forwarding*, an important feature of firewalls that allows to redirect the incoming traffic towards the appropriate server, which is supposed to be in a specific zone of the network (unknown to anyone located outside). The application of a `reject` rule is similar: the incoming object is removed from the system and a new special object, representing an ICMP error packet, is created. The source and destination IP addresses of the new packet are exchanged with respect to the packet given as input, so that the new packet goes back to the sender to signal that its previous IP packet has been rejected. The destination port is set equal to the source port of the original packet, whereas the source port is simply put to 0. Since all these fields of the resulting packet are so determined, when writing a `reject` rule the only argument of *out_fields* which is meaningful is *gateway*, which indicates the direction to be followed to go back to the sender; all the other fields are ignored. The application of a `drop` rule simply removes the object from the system, without producing any new object. In this case, the argument *out_fields* can be omitted.

When a new object has been created as the result of the application of a rule, it is put in the region enclosed by the membrane, ready to be processed by the next list of rules. Such a list is located at the end of the edge which is uniquely determined by the *gateway* parameter; the presence of this field thus allows to simulate also *routing tables*, and is particularly useful when two regions are connected by two or more edges. Of course there may already be other objects waiting to be matched against these rules; at the next computation step, one of these objects will be chosen in a nondeterministic way and will be processed, according to the *sequential* mode of applying the rules. Since no object may be processed by two or more lists of rules in the same computation step, each list can be operated *in parallel* with the others. This situation is similar to what happens with spiking neural P systems [8], where each neuron applies its rules in the sequential way but all neurons work in parallel.

We conclude this section by giving the formal definition of our model of P systems. A *firewall P system*, of degree $m \geq 1$, is a tuple $\Pi = (O, Z_1, \ldots, Z_m, conn)$, where:

- O is the set of *objects*, which represent IP packets as described above;
- $Z_1, \ldots, Z_m$ are the *membranes* (cells) of the system, each representing a zone of the network. Each membrane is a tuple $Z_i = (w_i, L_{i_1}, \ldots, L_{i_k})$, where w_i is the multiset of objects initially present in the membrane, and k is the number of incoming edges (in-degree) of Z_i. For every $j \in \{1, 2, \ldots, k\}$, L_{i_j} is a set of

rules, associated to Z_i and to the j-th incoming edge, having the form described above:
- ($\mathit{priority}$, `accept`, *in_fields, out_fields*)
- ($\mathit{priority}$, `reject`, *in_fields, out_fields*)
- ($\mathit{priority}$, `drop`, *in_fields*)

- $\mathit{conn} = \big\{\{i,j\} : i,j \in \{1,2,\ldots,m\} \text{ and } i \neq j\big\}$ is the multigraph (that is, a multiset of edges) of *connections* between the membranes.

A *configuration* of a firewall P system Π is described by the multisets of objects contained in its membranes. The *initial configuration* is the one in which membrane Z_i contains the multiset w_i, for all $i \in \{1,2,\ldots,m\}$. A *computation step* changes the current configuration by applying the rules as described above. In particular, we recall that at every computation step each list of rules chooses in a nondeterministic way an object among those which have to be processed by such rules. Moreover, the lists operate with *maximal parallelism*: if at least one object exists which has to be processed by a given list of rules, then one of these objects *must* be chosen and matched against the rules. A *final configuration* is a configuration in which no rule can be applied. As usual, a *computation* starts from an initial configuration and produces configurations by applying computation steps. The computation *halts* if it reaches a final configuration. By identifying an *input membrane* Z_{in} and an *output membrane* Z_{out}, with $in, out \in \{1,2,\ldots,m\}$, we can define a computation device that transforms input multisets into output multisets: the input of the computation is w_{in}, whereas the output is the contents of membrane Z_{out} in the final configuration, if it is reached. Non-halting computations produce no output. By considering the Parikh vectors associated with multisets, we immediately obtain also a computation device that transforms vectors of natural numbers into vectors of natural numbers.

4 One Membrane Suffices

Let us give now some insight on the computational power of firewall P systems.

As stated in the Introduction, we envision that our systems will be used to mathematically prove some security properties. Such proofs will be obtained by considering *reachability* problems. So, for example, we could prove that a certain kind of TCP packets will never reach a specified zone of the network by showing that no configuration which can be reached from the initial configuration contains that kind of packets in that zone. This means that our P systems should *not* be Turing-complete, otherwise the reachability problem would be undecidable. However, it is easily proved that firewall P systems are not universal: since objects are never created (rules can only modify an object or remove it from the system, and no object can enter the system from the environment during the computation, as it happens with tissue P systems), no output which contains more objects than those given in the initial configuration may be produced.

Establishing the precise computational power of firewall P systems is left as an open problem. In this section we just prove that firewall P systems can be simulated by single-membrane transition P systems using non-cooperative rewriting rules with priorities and catalysts. If the simulated firewall P system would operate in the *sequential* mode (meaning that lists of rules do *not* work in the maximally parallel way) this would mean that they would generate at most the Parikh images of ET0L languages [18]. However, in order to correctly simulate the maximally parallel application of the lists of rules — while maintaining sequentiality between the rules of the same list — we have to use a catalyst for each list of rules.

Theorem 1. *Firewall P systems can be simulated by single-membrane transition P systems using non-cooperative rewriting rules with priorities and catalysts.*

Proof. Let $\Pi = (O, Z_1, \ldots, Z_m, conn)$ be a firewall P system, where $Z_i = (w_i, L_{i_1}, \ldots, L_{i_k})$. We build a transition P system $\Pi' = (A, \mu, w, R)$ that simulates Π as follows. The alphabet A is composed of objects which can be seen as seven-tuples

$$(\textit{region, protocol, src\_IP, src\_port, dst\_IP, dst\_port, gateway})$$

where (*protocol, src_IP, src_port, dst_IP, dst_port, gateway*) is an object of Π, and *region* keeps track of the region which contains the object. Moreover, alphabet A contains also a symbol $\sharp \notin O$, and a symbol $c_{i,j} \notin O$ for each set of rules L_j of simulated membrane Z_i. The membrane structure μ is composed by a single membrane, the skin. The multiset w of the objects initially present in the skin membrane is built from the multisets w_i by adding to each object $o \in w_i$ the new value of the *region* component (that, for w_i, is equal to i). This initial multiset also contains a single copy of each symbol $c_{i,j} \in A$.

The set R of rewriting rules is obtained from the lists of rules of Π as follows. Let $r_{i,j} = (\textit{priority}, \texttt{accept}, \textit{in\_fields}, \textit{out\_fields})$ be an `accept` rule of Π, associated with membrane i and its incoming edge j. This rule produces a set $R_{i,j}$ of rewriting rules in Π', where each rule is obtained by specifying a source and a destination IP address, taken from the subnets specified in *in_fields* in all possible ways. For every pair (*src_IP, dst_IP*) of IP addresses a rule $ac_{i,j} \to bc_{i,j} \in R_{i,j}$ is generated, where a is the object that represents the IP packet being analyzed, and b represents the packet modified according with the values of *out_fields*. Object $c_{i,j}$ is a catalyst, unique to membrane i and its incoming edge j, which is used to make the application of rules from the same list sequential; indeed, for all i and j, system Π' contains a single copy of $c_{i,j}$. The value of *region* in b is i, while the value of *region* in a is put equal to the membrane which is connected to membrane i through its incoming edge j. The priority of the rule is set equal to the value of *priority* from $r_{i,j}$. Each `reject` rule produces an analogous set of rewriting rules, whereas all `drop` rules generate rewriting rules in which the output object is $\sharp$, that does not appear in the left hand side of any rule. The set R of rules is obtained as the union of all sets $R_{i,j}$ thus generated.

The system Π' thus generated operates in the maximally parallel way. Due to the presence of catalysts, the rules occurring in the same list of Π are simulated by Π' in the sequential way. It is not difficult to see that each computation step of Π corresponds to a computation step of Π', and hence that Π' simulates Π. □

Please note that in firewall P systems we can have rules with wildcard patterns when the fields *src_IP/subnet, dst_IP/subnet* have a subfield *subnet* of value less than 32, or anywhere a value `any` is used in a field, but these wildcards will have matches only over a predefined finite set of values (valid IP addresses, protocols, ...). Ours is therefore just a syntactic short notation for finite sets of usual P rules, and this comes into play also in the previous proof, when building the set of rules of the simulating transition P system.

Remark.

The possibility of modeling a network of firewalls by an equivalent single formal element has been studied also in a completely different technical context, when using SAT instances as a representation of the system [9]. Moreover, this property suggests the interesting perspective of having actual network systems where all the subnets are seen as a single region. In the actual implementation of such a system, we could consider to write in the header of IP packets the value of the *region* component used by our single membrane model, and anywhere a packet would arrive, it would be filtered/transformed by active elements of the network. This technological approach could enhance network security, since filtering would no longer happen only in firewalls on the borders, with the trouble of them being "single points of failure", but consistently anywhere on the network.

5 Conclusions and Directions for Future Work

In this paper we have modeled the functioning of firewalls, routers, and other simple rule filters by a tissue-like model of P systems. After a description of the features of this model, we have formally defined it and we have shown that it can be easily simulated by a single-membrane P system. We can thus argue that what is seen as an important and difficult problem in the practice of computer networks (analyzing and understanding long lists of filtering rules) is indeed a very simple task in the theory of membrane systems.

Future work includes further analysis of the features of packet filters and of the properties of the corresponding P systems. In particular, it should be interesting to see how the algorithms currently proposed in the literature (such as those in [3, 4, 5]) to solve common firewall misconfiguration problems map themselves to the model of P systems we have proposed. Also some new algorithms may be devised, based upon the new point of view given by P systems. Since many simulators of P systems already exist, an interesting development is to test the application of P systems described in the current paper against real cases.

Acknowledgements. This work was partially supported by the Italian project FIAR 2008 "Modelli di calcolo naturale e applicazioni alla systems biology".

References

1. P. Eronen, J. Zitting. An expert system for analyzing firewall rules. In *Proceedings of the* 6th *Nordic Workshop on Secure IT Systems (NordSec 2001)*, 2001, pp. 100–107.
2. J.D. Guttman. Filtering postures: local enforcement for global policies. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.
3. A. Hari, S. Suri, G. Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings of IEEE INFOCOM 2000*, 2000, pp. 1203–1212.
4. S. Hazelhurst. Algorithms for analysing firewall and router access lists. *Technical Report TR-Wits-CS-1999-5*, Department of Computer science, University of the Witwatersrand, South Africa, 1999.
5. S. Hazelhurst, A. Attar, R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000)*, IEEE Computer Society Press, 2000, pp. 576–585.
6. S. Hazelhurst, A. Fatti, A. Henwood. Binary decision diagram representations of firewall and router access lists. *Technical Report TR-Wits-CS-1998-3*, Department of Computer Science, University of Witwatersrand, South Africa, 1998.
7. K. Hundley, G. Held. *Cisco access lists field guide.* McGraw-Hill, 2000.
8. M. Ionescu, Gh. Păun, T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae* **71**(2-3):279–308, 2006.
9. A. Jeffrey, T. Samak. Model checking firewall policy configurations. In *Proc. of the 2009 IEEE Symposium on Policies for Distributed Systems and Networks*, 2009, pp. 60–67.
10. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón. A new class of symbolic abstract neural nets: tissue P systems. *Computing and Combinatorics,* 8th *Annual International Conference, COCOON 2002*, LNCS 2387, Springer, Berlin, 2002, pp. 290–299.
11. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón. Tissue P systems. *Theoretical Computer Science*, **296**(2):295–326, 2003.
12. A. Mayer, A. Wool, E. Ziskind. Fang: A firewall analysis engine. In *Proc. of the 2000 IEEE Symposium on Security and Privacy*, 2000, pp. 177–187.
13. The NETFILTER/IPCHAINS/IPTABLES web page: `http://www.netfilter.org/`
14. The P systems web page: `http://ppage.psystems.eu/`
15. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**:108–143, 2000.
16. Gh. Păun. *Membrane computing. An introduction.* Springer, 2002.
17. Gh. Păun, G. Rozenberg. An introduction to and an overview of membrane computing. In Gh. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook Of Membrane Computing*, Oxford University Press, 2010, pp. 1–27.
18. D. Sburlan. Non-cooperative P systems with priorities characterize PsET0L. In *Membrane Computing,* 6th *International Workshop, WMC 2005*, LNCS 3850, Springer, 2006, pp. 363–370.
19. A.S. Tanenbaum. *Computer networks.* Prentice Hall, 2002.

Gandy-Păun-Rozenberg Machines

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences,
Śniadeckich 8, P.O.Box 21, 00-956 Warsaw, Poland
adamo@impan.gov.pl

Mathematics is a powerful tool that helps people achieve new goals as well as understand what is impossible to do.
Mark Burgin

Summary. Gandy–Păun–Rozenberg machines are introduced as certain graph rewriting systems. A representation of Gandy–Păun–Rozenberg machines by Gandy machines is given. A construction of a Gandy–Păun–Rozenberg machine solving 3-SAT problem in a polynomial time is shown.

1 Introduction

The paper [8] by Eric Steinhart contains a discussion of logical foundations of computation theory including quantum computing which gives rise to the following family of questions:
(?) what is it an $\mathcal{X}$ possible machine?
for $\mathcal{X} \in$ {set-theoretically, discrete topologically, continuous topologically, geometrically, biologically inspired, physically, cognitive and intelligent}.

We point out here that Robin Gandy's machines (cf. Gandy's paper [1]) yield some answer to (?) for $\mathcal{X} \equiv$ set-theoretically in discrete case. The physically possible machines are discussed in the papers about physical limitations of computing devices by Scott Aaronson, Jacob Bekenstein, Charles H. Bennett, Rolf Landauer, Stockmeyer and Meyer, among others.

The paper [9] by Jiří Wiedermann inspired to formulate (?) for $\mathcal{X} \equiv$ cognitive and intelligent.

An idea of a Gandy–Păun–Rozenberg machine, briefly G–P–R machine, introduced in Section 2, is aimed to provide an answer to (?) for $\mathcal{X} \equiv$ set-theoretically, $\mathcal{X} \equiv$ discrete topologically, and $\mathcal{X} \equiv$ biologically inspired.

The G–P–R machines are the constructs which have common features with or are related to:

— Gandy's machines,
— P systems due to Gheorghe Păun (cf. [6]),
— parallel rewriting systems of graphs investigated by Grzegorz Rozenberg himself with scientists cooperating with him, among others, in preparation and editing of many volume *Handbook of graph grammars and computing by graph transformation* [2].

The core of a G–P–R machine is a finite set of rewriting rules for certain finite directed labelled graphs, where these graphs are instantenous descriptions for the computation process realized by the machine.

The conflictless parallel (simultaneous) application of the rewriting rules of a G–P–R machine is realized in Gandy's machine mode (according to Local Causality Principle), where (local) maximality of "causal neighbourhoods" replaces (global) maximality of, e.g. conflictless set of evolution rules applied simultaneously to a membrane structure which appears during the evolution process generated by a P system. Therefore one can construct a Gandy's machine from a G–P–R machine in an immediate way, see Section 2.

The NP complete problems can be solved by G–P–R machines in a polynomial time (but with an exponential number of indecomposable processors), see Section 3, where we construct a G–P–R machines solving SAT problem in a polynomial time in a similar way to (families of) P systems solving this problem also in a polynomial time (cf. the pioneering Păun's paper [5]).

Randomized G–P–R machines for solving NP problems in a polynomial time with subexponential number of indecomposable processors are forthcoming.

An extension of G–P–R machines to the case of cellular automata can be done by adopting the idea of cellular hypergraph rewriting introduced by Peter Hartmann in his paper [3].

2 Gandy-Păun-Rozenberg machines and Gandy machines

For all unexplained terms and notation of category theory and graph theory we refer the reader to Appendix.

Definition. A G–P–R *machine* $\mathcal{M}$ is determined by the following data:

— a finite set $\Sigma_{\mathcal{M}}$ of labels or symbols of $\mathcal{M}$,
— a skeletal set $\mathcal{S}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over Σ, which are called *instantenous descriptions* of $\mathcal{M}$,
— a function $\mathcal{F}_{\mathcal{M}} : \mathcal{S}_{\mathcal{M}} \to \mathcal{S}_{\mathcal{M}}$ called the *transition function* of $\mathcal{M}$,
— a function $\mathcal{R}_{\mathcal{M}}$: $\mathrm{PREM}_{\mathcal{M}} \to \mathrm{CONCL}_{\mathcal{M}}$ from a finite skeletal set $\mathrm{PREM}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ onto a finite skeletal set $\mathrm{CONCL}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ such that $\mathcal{R}_{\mathcal{M}}$ determines the *set*

$$\widetilde{\mathcal{R}}_{\mathcal{M}} = \{P \vdash C \mid P \in \mathrm{PREM}_{\mathcal{M}} \text{ and } C = \mathcal{R}_{\mathcal{M}}(P)\}$$

of rewriting rules of $\mathcal{M}$ which are identified with ordered pairs $r = (P_r, C_r)$, where the graph $P_r \in \mathrm{PREM}_{\mathcal{M}}$ is the *premise* of r and the graph $C_r = \mathcal{R}_{\mathcal{M}}(P_r)$ is the *conclusion* of r,

— a subset $\mathcal{I}_{\mathcal{M}}$ of $\mathcal{S}_{\mathcal{M}}$ which is the set of *initial instantaneous descriptions of* $\mathcal{M}$.

The above data are subject of the following conditions:

1) $V(\mathcal{G}) \subseteq V(\mathcal{F}_{\mathcal{M}}(\mathcal{G}))$ for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$,
2) $V(\mathcal{G}) \subseteq V(\mathcal{R}_{\mathcal{M}}(\mathcal{G}))$ for every $\mathcal{G} \in \mathrm{PREM}_{\mathcal{M}}$,
3) the rewriting rules of $\mathcal{M}$ are *applicable* to $\mathcal{S}_{\mathcal{M}}$ which means that for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$ the set

$$\begin{aligned}\mathcal{P}\ell(\mathcal{G}) = \big\{h \mid\ & h \text{ is an embedding of labelled graphs over } \Sigma\\ & \text{with } \mathrm{dom}(h) \in \mathrm{PREM}_{\mathcal{M}} \text{ and } \mathrm{cod}(h) = \mathcal{G}\\ & \text{such that for every embedding } h' \text{ of labelled graphs over } \Sigma\\ & \text{with } \mathrm{dom}(h') \in \mathrm{PREM}_{\mathcal{M}} \text{ and } \mathrm{cod}(h') = \mathcal{G}\\ & \text{if } \mathrm{im}(h) \text{ is a labelled subgraph of } \mathrm{im}(h'), \text{ then } h = h'\big\}\end{aligned}$$

of *maximal applications* h of the rules $\mathrm{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h))$ of $\mathcal{M}$ in places $\mathrm{im}(h)$ is such that the following conditions hold:

(i) $V(\mathcal{G}) = \bigcup_{h \in \mathcal{P}\ell(\mathcal{G})} V(\mathrm{im}(h))$, $E(\mathcal{G}) = \bigcup_{h \in \mathcal{P}\ell(\mathcal{G})} E(\mathrm{im}(h))$,

(ii) for all $h_1, h_2 \in \mathcal{P}\ell(\mathcal{G})$ the equation $\ell_{\mathcal{G}_{h_1}}(\dot{h}_1^{-1}(v)) = \ell_{\mathcal{G}_{h_2}}(\dot{h}_2^{-1}(v))$ holds for every $v \in V(\mathrm{im}(h_1)) \cap V(\mathrm{im}(h_2))$, where $\ell_{\mathcal{G}_{h_1}}$, $\ell_{\mathcal{G}_{h_2}}$ are the labelling functions of $\mathcal{G}_{h_1} = \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h_1))$, $\mathcal{G}_{h_2} = \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h_2))$, respectively, and $\dot{h}_1^{-1}$, $\dot{h}_2^{-1}$ are the inverses of isomorphisms induced by the embeddings h_1, h_2, respectively.

(iii) $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is a colimit of a gluing diagram $\mathcal{D}^{\mathcal{G}}$ constructed in the following way (the construction of $\mathcal{D}^{\mathcal{G}}$ is provided by (ii)):

- the set $\mathcal{I}$ of indexes of $\mathcal{D}^{\mathcal{G}}$ is such that $\mathcal{I} = \mathcal{P}\ell(\mathcal{G}) \cup \{\Delta\}$, where $\Delta \notin \mathcal{P}\ell(\mathcal{G})$ is the center of $\mathcal{D}^{\mathcal{G}}$,
- the family $\mathcal{G}_i$ $(i \in \mathcal{I})$ of labelled graphs of $\mathcal{D}^{\mathcal{G}}$ is such that $\mathcal{G}_h = \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h))$ for every $h \in \mathcal{P}\ell(\mathcal{G})$, and $\mathcal{G}_\Delta$ is such that $V(\mathcal{G}_\Delta) = V(\mathcal{G})$, $E(\mathcal{G}_\Delta) = \varnothing$, and the labelling function $\ell_{\mathcal{G}_\Delta}$ is such that provided by (ii)
$$\ell_{\mathcal{G}_\Delta}(v) = \ell_{\mathcal{G}_h}(\dot{h}^{-1}(v))$$
for every $v \in V(\mathrm{im}(h))$ and every $h \in \mathcal{P}\ell(\mathcal{G})$, where $\dot{h}^{-1}$ is the inverse of the isomorphism $\dot{h}$ induced by the embedding h,
- the gluing conditions gl_h $(h \in \mathcal{P}\ell(\mathcal{G}))$ of $\mathcal{D}^{\mathcal{G}}$ are defined by
$$\mathrm{gl}_h = \big\{(v, \dot{h}^{-1}(v)) \mid v \in V(\mathrm{im}(h))\big\}$$
for every $h \in \mathcal{P}\ell(\mathcal{G})$, where $\dot{h}^{-1}$ is the inverse of the isomorphism $\dot{h}$ induced by embedding h,

(iv) the following equations hold:

$$\begin{aligned} V(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) &= \bigcup_{i\in\mathcal{I}} V(\mathrm{im}(q_i)) \\ \text{and } E(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) &= \bigcup_{i\in\mathcal{I}} E(\mathrm{im}(q_i)) \end{aligned}$$

for the canonical injections $q_i : \mathcal{G}_i \to \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ $(i \in \mathcal{I})$ forming a colimiting cocone of the diagram $\mathcal{D}^{\mathcal{G}}$ defined in (iii),

(v) the canonical injection $q_{\Delta} : \mathcal{G}_{\Delta} \to \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is an inclusion of labelled graphs, where Δ is the center of $\mathcal{D}^{\mathcal{G}}$ and q_{Δ} is an element of the colimiting cocone in (iv).

Thus $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is the result of simultaneous application of the rules $\mathrm{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h))$ in the places $\mathrm{im}(h)$ for $h \in \mathcal{P}\ell(\mathcal{G})$, where one replaces simultaneously $\mathrm{im}(h)$ by $\mathrm{im}(q_h)$ in $\mathcal{G}$ for $h \in \mathcal{P}\ell(\mathcal{G})$, respectively.

A finite sequence $\big(\mathcal{F}^i_{\mathcal{M}}(\mathcal{G})\big)_{i=0}^{n}$ is called a *finite computation of* $\mathcal{M}$, the number n is called the *time* of this computation, and $\mathcal{F}^n_{\mathcal{M}}(\mathcal{G})$ is called the *final instantaneous description* for this computation if

$$\mathcal{F}^0_{\mathcal{M}}(\mathcal{G}) = \mathcal{G} \in \mathcal{I}_{\mathcal{M}}, \quad \mathcal{F}^{n-1}_{\mathcal{M}}(\mathcal{G}) \neq \mathcal{F}^n_{\mathcal{M}}(\mathcal{G}), \quad \text{and } \mathcal{F}_{\mathcal{M}}(\mathcal{F}^n_{\mathcal{M}}(\mathcal{G})) = \mathcal{F}^n_{\mathcal{M}}(\mathcal{G}),$$

where $\mathcal{F}^i_{\mathcal{M}}(\mathcal{G})$ is defined inductively: $\mathcal{F}^i_{\mathcal{M}}(\mathcal{G}) = \mathcal{F}_{\mathcal{M}}\big(\mathcal{F}^{i-1}_{\mathcal{M}}(\mathcal{G})\big)$. □

We introduce the following auxiliary constructs which will be used to define those Gandy machines which represent G–P–R machines. For all unexplained terms concerning Gandy machines and hereditarily finite sets we refer the reader to [1], [7].

If the set $\Sigma_{\mathcal{M}}$ of labels of a G–P–R machine is an m-element set, we choose a bijection $\nabla : \Sigma_{\mathcal{M}} \to \{1,\dots,m\}$ and an urelement u to code the labels $\sigma \in \Sigma_{\mathcal{M}}$ by hereditarily finite sets $\{u\}^{\nabla(\sigma)+1}$, where one defines $\{u\}^1 = \{u\}$, $\{u\}^{k+1} = \big\{\{u\}^k\big\}$ for a natural number $k > 0$.

Then for a labelled directed graph $\mathcal{G}$ belonging to the set $\mathcal{S}_{\mathcal{M}}$ of instantaneous descriptions of a G–P–R machine $\mathcal{M}$, an injection $\alpha : V(\mathcal{G}) \to U$ into the set U of urelements, and an urelement $u = \alpha(v)$ for some v one defines a hereditarily finite set

$$\begin{aligned} H(\alpha,u,\mathcal{G}) = \Big\{\big\{\alpha(v_1),\{\alpha(v_2)\}\big\} \;\Big|\; (v_1,v_2) \in E(\mathcal{G})\Big\} \\ \cup \Big\{\big\{\alpha(v),\{\{u\}^{\nabla(\ell_{\mathcal{G}}(v))+1}\}\big\} \;\Big|\; v \in V(\mathcal{G})\Big\}, \end{aligned}$$

where $\ell_{\mathcal{G}}$ is the labelling function of $\mathcal{G}$.

Since $\mathcal{S}_{\mathcal{M}}$ is a skeletal set of isomorphically perfect graphs, the assignment $H(\alpha,u,\mathcal{G})$ defined above is a bijection from

$$\begin{aligned} \mathcal{S}^+_{\mathcal{M}} = \big\{(\alpha,u,\mathcal{G}) \mid \mathcal{G} \in \mathcal{S}_{\mathcal{M}},\ \alpha : V(\mathcal{G}) \to U \text{ is an injection,} \\ \text{and } u = \alpha(v) \text{ for some } v\big\} \end{aligned}$$

into

$$\mathcal{S}^*_{\mathcal{M}} = \{H(\alpha, u, \mathcal{G}) \mid (\alpha, u, \mathcal{G}) \in \mathcal{S}^+_{\mathcal{M}}\},$$

where $\mathcal{S}^*_{\mathcal{M}}$ appears a structural set of hereditarily finite sets understood as in Gandy's paper [1]. We use this set $\mathcal{S}^*_{\mathcal{M}}$ as the set of state-descriptions of a Gandy machine aimed to represent a G–P–R machine $\mathcal{M}$.

Then we choose a mapping $F^+ : \mathcal{S}^+_{\mathcal{M}} \to \mathcal{S}^+_{\mathcal{M}}$ such that

$$F^+(\alpha, u, \mathcal{G}) = (\widehat{\alpha}, u, \mathcal{F}_{\mathcal{M}}(\mathcal{G}))$$

for the transition function $\mathcal{F}_{\mathcal{M}}$ of a G–P–R machine $\mathcal{M}$ and for a chosen injection $\widehat{\alpha} : V(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) \to U$ such that

$$\widehat{\alpha}(v) = \alpha(v) \text{ for every } v \in V(\mathcal{G}) \subseteq V(\mathcal{F}_{\mathcal{M}}(\mathcal{G})).$$

Then we define a mapping $\mathcal{F}^*_{\mathcal{M}} : \mathcal{S}^*_{\mathcal{M}} \to \mathcal{S}^*_{\mathcal{M}}$ such that

$$\mathcal{F}^*_{\mathcal{M}}\big(H(\alpha, u, \mathcal{G})\big) = H\big(F^+(\alpha, u, \mathcal{G})\big).$$

This mapping $\mathcal{F}^*_{\mathcal{M}}$ appears a structural mapping understood as in Gandy's paper [1] and we use it as the transition function of a Gandy machine aimed to represent a G–P–R machine $\mathcal{M}$ which is described in the following theorem.

Theorem 1 (Representation of G–P–R machines by Gandy machines). *Let $\mathcal{M}$ be a G–P–R machine. Then $\mathcal{M}$ determines a Gandy machine $\mathsf{G}_{\mathcal{M}}$ whose set of state-descriptions is $\mathcal{S}^*_{\mathcal{M}}$, the transition function of $\mathsf{G}_{\mathcal{M}}$ is $\mathcal{F}^*_{\mathcal{M}}$, the sets T_1, T_2 of stereotypes of $\mathsf{G}_{\mathcal{M}}$ and the structural functions G_1, G_2 of $\mathsf{G}_{\mathcal{M}}$ are such that*

$$T_1 = T_2 = \mathrm{PREM}^*_{\mathcal{M}}/\cong \quad \textit{and} \quad G_1 = G_2 = \mathcal{R}^*_{\mathcal{M}},$$

where $\mathrm{PREM}^*_{\mathcal{M}}$ is defined for $\mathrm{PREM}_{\mathcal{M}}$ in an analogous way as $\mathcal{S}^*_{\mathcal{M}}$ has been defined for $\mathcal{S}_{\mathcal{M}}$, $\mathrm{PREM}^*_{\mathcal{M}}/\cong$ is the set of equivalence classes with respect to isomorphism relation $\cong$ of hereditarily finite sets defined in Gandy's paper [1], *and $\mathcal{R}^*_{\mathcal{M}}$ is defined for $\mathcal{R}_{\mathcal{M}}$ in an analogous way as $\mathcal{F}^*_{\mathcal{M}}$ has been defined for $\mathcal{F}_{\mathcal{M}}$.*

Proof. The assumption that $\mathcal{F}(\mathcal{G})$ is a colimit of the gluing diagram $\mathcal{D}^{\mathcal{G}}$ and Lemma 5 in the Appendix provide that the conditions $(3)_r$ of Principle IV in Gandy's paper [1] hold for $\mathsf{G}_{\mathcal{M}}$. □

The assignment $H(\alpha, u, \mathcal{G})$ and then the definition of $\mathcal{S}^*_{\mathcal{M}}$ were inspired by the similar constructions in [7].

The examples of G–P–R machines are presented in the next section.

3 Gandy-Păun-Rozenberg machines and NP complete problems

We show a construction of a G–P–R machine which solves NP complete 3-SAT problem in a polynomial time. We begin with presentation of examples of those

G–P–R machines which simulate the computations of Turing machines and the computations of certain Boolean circuits, respectively, and which are used in the construction.

For all unexplained terms of logic and computational complexity theory, including Turing machines and the formulation of SAT and 3-SAT problems, we refer the reader to [4].

We use the following two types of labelled directed graphs.

Definitions. We say that an ordered triple (k, m, n) of integers k, m, n is *acceptable* if $k > 0$, $m \neq 0$, $n > 1$, and $-k < m < n$. We define

$$\mathrm{lin}[k,n] = \big\{(i, i+1) \mid i \text{ is an integer such that } -k \leq i < -1 \text{ or } 1 \leq i \leq n\big\} \\ \cup \{(-1,1)\}$$

for k, n as above.

Then we say that a labelled directed graph $\mathcal{G}$ over Σ having more than one label is *induced by an acceptable ordered triple* (k, m, n) if $\mathcal{G}$ is such that

— $V(\mathcal{G}) = \big\{i \mid i \text{ is an integer such that } -k \leq i \leq n\big\}$,
— $E(\mathcal{G}) = \mathrm{lin}[k,n] \cup \{(0,m),(1,1)\}$,
— $\ell_{\mathcal{G}}(0) \notin \{\ell_{\mathcal{G}}(k), \ell_{\mathcal{G}}(m)\}$.

For a natural number $n > 0$ a *regular labelled binary tree of depth n over* $\{\mathrm{root}, 0, 1\} \times \Sigma$ is defined to be a labelled directed graph $\mathcal{T}$ over $\{\mathrm{root}, 0, 1\} \times \Sigma$ such that

— $V(\mathcal{T})$ is the set of binary strings[1] of length not greater than n including empty string Λ,
— $E(\mathcal{T}) = \big\{(\Gamma, \Gamma i) \mid \{\Gamma, \Gamma i\} \subseteq V(\mathcal{T}) \text{ and } i \in \{0,1\}\big\}$
$\cup \big\{(\Gamma, \Gamma) \mid \Gamma \text{ is a binary string of length } n\big\}$,
— the labelling function $\ell_{\mathcal{T}} : V(\mathcal{T}) \to \{\mathrm{root}, 0, 1\} \times \Sigma$ of $\mathcal{T}$ is such that $\ell^1_{\mathcal{T}}(\Lambda) = \mathrm{root}$, $\ell^1_{\mathcal{T}}(\Gamma i) = i$ for every binary string Γ and every $i \in \{0,1\}$ such that $\Gamma i \in V(\mathcal{T})$,

where $\ell^1_{\mathcal{T}}(x)$, $\ell^2_{\mathcal{T}}(x)$ denote the coordinates such that $\ell_{\mathcal{T}}(x) = (\ell^1_{\mathcal{T}}(x), \ell^2_{\mathcal{T}}(x))$ and Γi denotes that binary string Θ whose last element is the digit i, and Γ is that binary string which is the result of deleting the last element in Θ.

Lemma 1. *The set of labelled directed graphs over Σ induced by acceptable ordered triples of integers is a skeletal set of isomorphically perfect graphs for Σ having more than one label.*

Lemma 2. *The set of all regular binary trees of arbitrary depth over $\{\mathrm{root}, 0, 1\} \times \Sigma$ is a skeletal set of isomorphically perfect graphs.*

[1] A binary string is a sequence, maybe empty, of digits 0, 1.

Example 1 (G–P–R machine simulating the computations of a Turing machine). Let $\mathbb{T}$ be a Turing machine whose alphabet Σ (including blank symbol) is disjoint with the set Q of states of $\mathbb{T}$ and let $\delta : \Sigma \times Q \to \Sigma \times Q \times \{L, 0, R\}$ be the transition of $\mathbb{T}$ with cursor directions L for "left", 0 for "stay", and R for "right". We define a *graphical instantaneous description of* $\mathbb{T}$ to be a labelled directed graph $\mathcal{G}$ over $\Sigma^0 = \Sigma \cup Q \cup \{\%, \S\}$ with $\{\%, \S\} \cap (\Sigma \cup Q) = \varnothing$ such that

— $\mathcal{G}$ is induced by some acceptable ordered triple of integers,
— if $\mathcal{G}$ is induced by an acceptable ordered triple (k, m, n) of integers, then $\ell_{\mathcal{G}}(-k) = \%$, $\ell_{\mathcal{G}}(0) \in Q$, $\ell_{\mathcal{G}}(n) = \S$ and $\ell_{\mathcal{G}}(j) \in \Sigma$ for every $j \in \big\{i \in V(\mathcal{G}) \mid -k < i < n \text{ and } i \neq 0\big\}$ (here m corresponds to cursor position on Turing machine tape indicated by the edge $(0, m)$).

By Lemma 1 the set $\mathcal{S}_{\mathbb{T}}$ of all graphical instantaneous descriptions of $\mathbb{T}$ is a skeletal set of isomorphically perfect labelled graphs. Thus we define a G–P–R machine $\mathcal{M}_{\mathbb{T}}$ aimed to simulate the computations of $\mathbb{T}$ such that

— the set of instantaneous descriptions of $\mathcal{M}_{\mathbb{T}}$ is the set $\mathcal{S}_{\mathbb{T}}$ of graphical instantaneous descriptions of $\mathbb{T}$,
— the transition function $\mathcal{F}_{\mathbb{T}}$ of $\mathcal{M}_{\mathbb{T}}$ and the rewriting rules of $\mathcal{M}_{\mathbb{T}}$ are determined by the transition function δ of $\mathbb{T}$ such that if $\delta(a, q) = (a', q', R)$, then
  (f^R) if $\mathcal{G} \in \mathcal{S}_{\mathbb{T}}$ and $\mathcal{G}$ is induced by (k, m, n) such that $\ell_{\mathcal{G}}(m) = a$, $\ell_{\mathcal{G}}(0) = q$, then
    (f^R_1) if $m < n - 1$ then $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ is that $\mathcal{G}'$ which is induced by $(k, \widehat{m}, n)$ with $\widehat{m} = m + 1$ for $m \neq -1$ and $m = 1$ for $m = -1$ such that $\ell_{\mathcal{G}'}(0) = q'$, $\ell_{\mathcal{G}'}(m) = a'$, and $\ell_{\mathcal{G}'}(i) = \ell_{\mathcal{G}}(i)$ for every $i \in V(\mathcal{G}) - \{0, m\}$,
    (f^R_2) if $m = n - 1$ then $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ is that $\mathcal{G}'$ which is induced by $(k, m+1, n+1)$ such that $\ell_{\mathcal{G}'}(0) = q'$, $\ell_{\mathcal{G}'}(m) = a'$, $\ell_{\mathcal{G}'}(n)$ is blank symbol, and $\ell_{\mathcal{G}'}(i) = \ell_{\mathcal{G}}(i)$ for every $i \in V(\mathcal{G}') - \{0, m, n\}$,
  (r^R) the rewriting rules are given by the following two schemes $\mathcal{G}_p \vdash \mathcal{G}_c$ such that
    (r^R_1) the premise $\mathcal{G}_p$ is such that $V(\mathcal{G}_p) = \{-1, 0, 1, 2\}$, $E(\mathcal{G}_p) = \mathrm{lin}[1, 2] \cup \{(0, 1)\}$, $\ell_{\mathcal{G}_p}(-1) \in \Sigma \cup \{\%\}$, $\ell_{\mathcal{G}_p}(0) = q$, $\ell_{\mathcal{G}_p}(1) = a$, $\ell_{\mathcal{G}_p}(2) \in \Sigma$,
    the conclusion $\mathcal{G}_c$ is such that $V(\mathcal{G}_c) = V(\mathcal{G}_p)$, $E(\mathcal{G}_c) = \mathrm{lin}[1, 2] \cup \{(0, 2)\}$, $\ell_{\mathcal{G}_c}(-1) = \ell_{\mathcal{G}_p}(-1)$, $\ell_{\mathcal{G}_c}(0) = q'$, $\ell_{\mathcal{G}_c}(1) = a'$, and $\ell_{\mathcal{G}_c}(2) = \ell_{\mathcal{G}_p}(2)$.
    (r^R_2) the premise $\mathcal{G}_p$ is such that $V(\mathcal{G}_p) = \{-1, 0, 1, 2\}$, $E(\mathcal{G}_p) = \mathrm{lin}[1, 2] \cup \{(0, 1)\}$, $\ell_{\mathcal{G}_p}(-1) \in \Sigma \cup \{\%\}$, $\ell_{\mathcal{G}_p}(0) = q$, $\ell_{\mathcal{G}_p}(1) = a$, $\ell_{\mathcal{G}_p}(2) = \S$,
    the conclusion $\mathcal{G}_c$ is such that $V(\mathcal{G}_c) = \{-1, 0, 1, 2, 3\}$, $E(\mathcal{G}_c) = \mathrm{lin}[1, 3] \cup \{(0, 2)\}$, $\ell_{\mathcal{G}_c}(-1) = \ell_{\mathcal{G}_p}(-1)$, $\ell_{\mathcal{G}_c}(0) = q'$, $\ell_{\mathcal{G}_c}(1) = a'$, $\ell_{\mathcal{G}_c}(2)$ is blank symbol, and $\ell_{\mathcal{G}_c}(3) = \S$.

For the cases of equations $\delta(a, q) = (a', q', 0)$ and $\delta(a, q) = (a', q', L)$ the values $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ and the rewriting rules are defined in a similar way, where, e.g., the counterpart of (f^R_2) for $\delta(a, q) = (a', q', L)$ is:

(f^L_2) if $1 = m = k$ or $-k + 1 = m \neq 0$, then $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ is that $\mathcal{G}'$ which is induced by $(k + 1, -k, n)$ such that $\ell_{\mathcal{G}'}(-k - 1) = \%$, $\ell_{\mathcal{G}'}(-k)$ is blank symbol,

$\ell_{\mathcal{G}'}(0) = q'$, $\ell_{\mathcal{G}'}(m) = a'$, and $\ell_{\mathcal{G}'}(i) = \ell_{\mathcal{G}}(i)$ for all $i \in V(\mathcal{G}') - \{-k-1, -k, 0, m\}$.

The versions of the above rules $\mathcal{G}_p \vdash \mathcal{G}_c$ for both $\mathcal{G}_p$ and $\mathcal{G}_c$ completed by the loop (i,i) for a unique $i \in V(\mathcal{G}_p)$ with $\ell_{\mathcal{G}_p}(i) \notin \{\%, \S\} \cup Q$ are also necessary. The identity rules $\mathcal{G} \vdash \mathcal{G}$ are also necessary, where $\mathcal{G}$ is of the following two forms:

(id$_1$) $V(\mathcal{G}) = \{0,1\}$, $E(\mathcal{G}) = \{(0,1)\}$, $\{\ell_{\mathcal{G}}(0), \ell_{\mathcal{G}}(1)\} \subset \Sigma^0 - Q$,
(id$_2$) $V(\mathcal{G}) = \{0\}$, $E(\mathcal{G}) = \{(0,0)\}$, $\ell_{\mathcal{G}}(0) \in \Sigma$.

There is no other rewriting rule of $\mathcal{M}_{\mathbb{T}}$ than that described by the above schemes.

Since the graphical instantaneous descriptions of a Turing machine $\mathbb{T}$ coincide with the usual instantaneous descriptions of $\mathbb{T}$ or configurations of $\mathbb{T}$ as in [4], the G–P–R machine $\mathcal{M}_{\mathbb{T}}$ simulates the computations of $\mathbb{T}$ due to definition of $\mathcal{F}_{\mathbb{T}}$.

Example 2 (G–P–R machine simulating the computations of certain Boolean circuits). We define a *disjunctive circuit* G–P–R *machine* $\mathcal{M}_{\text{circ}}$ which is aimed to simulate computations of certain tree like Boolean circuits such that

— the set $\mathcal{S}_{\text{circ}}$ of instantaneous descriptions of $\mathcal{M}_{\text{circ}}$ is the set of those regular labelled binary trees $\mathcal{T}$ of depth greater than 3 over the set $\{\text{root}, 0, 1\} \times \{\perp, 0, 1\}$ of labels which satisfy the following condition

(circ$_0$) for every binary string $\Gamma \in V(\mathcal{T})$ of length equal to the depth of $\mathcal{T}$ the number of elements of the set

$$\big\{i \mid i \text{ is a natural number with } 0 < i \leq n \text{ such that } \ell^2_{\mathcal{T}}(\Gamma \restriction i) \neq \perp\big\}$$

is not greater than 1 (thus this set may be empty), where n is the depth of $\mathcal{T}$ and if Γ is $(k_j)_{j=1}^{n}$ then $\Gamma \restriction i$ denotes the string $(k_j)_{j=1}^{i}$ which is Γ itself for $i = n$ and for $i < n$ $(k_j)_{j=1}^{i}$ is a shortening of Γ by cancellation of the elements $k_n, k_{n-1}, \ldots, k_{i+1}$.

— the transition function $\mathcal{F}_{\text{circ}}$ of $\mathcal{M}_{\text{circ}}$ is such that $\mathcal{F}_{\text{circ}}(\mathcal{T})$ is the result of simultaneous application to $\mathcal{T}$ in G–P–R machine mode the rewriting rules of $\mathcal{M}_{\text{circ}}$ which do not introduce new vertices and which are given by the following three schemes $\mathcal{T}_p \vdash \mathcal{T}_c$ such that

(circ$_1$) the premise $\mathcal{T}_p$ is such that $V(\mathcal{T}_p) = \{\Lambda, 0, 00, 01\}$,
$E(\mathcal{T}_p) = \big\{(\Lambda, 0), (0, 00), (0, 01), (00, 00), (01, 01)\big\}$,
$\ell^2_{\mathcal{T}_p}(\Lambda) = \ell^2_{\mathcal{T}_p}(0) = \perp$, $\{\ell^1_{\mathcal{T}_p}(\Lambda), \ell^1_{\mathcal{T}_p}(0)\} \subseteq \{0,1\}$,
$\ell^1_{\mathcal{T}_p}(00) = 0$, $\ell^1_{\mathcal{T}_p}(01) = 1$, $\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\} \subseteq \{0,1\}$,
the conclusion $\mathcal{T}_c$ is such that $V(\mathcal{T}_c) = V(\mathcal{T}_p)$, $E(\mathcal{T}_c) = E(\mathcal{T}_p)$, $\ell_{\mathcal{T}_c}(\Lambda) = \ell_{\mathcal{T}_p}(\Lambda)$, $\ell_{\mathcal{T}_c}(0) = \big(\ell^1_{\mathcal{T}_p}(0), \max\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\}\big)$,
$\ell_{\mathcal{T}_c}(00) = (0, \perp)$, $\ell_{\mathcal{T}_c}(01) = (1, \perp)$,

(circ$_2$) the premise $\mathcal{T}_p$ is such that $V(\mathcal{T}_p) = \{\Lambda, 0, 00, 01, 000, 001, 010, 011\}$,
$E(\mathcal{T}_p) = \big\{(\Gamma, \Gamma i) \mid \{\Gamma, \Gamma i\} \subseteq V(\mathcal{T}_p) \text{ and } i \in \{0,1\}\big\}$,
$\ell^2_{\mathcal{T}_p}(\Gamma) = \perp$ for all $\Gamma \in V(\mathcal{T}_p) - \{00, 01\}$,

$\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\} \subseteq \{0,1\}$, $\{\ell^1_{\mathcal{T}_p}(\Lambda), \ell^1_{\mathcal{T}_p}(0)\} \subseteq \{0,1\}$,
$\ell^1_{\mathcal{T}_p}(\Gamma i) = i$ for all $\Gamma \in \{0, 00, 01\}$ and $i \in \{0,1\}$,
the conclusion $\mathcal{T}_c$ is such that $V(\mathcal{T}_c) = V(\mathcal{T}_p)$, $E(\mathcal{T}_c) = E(\mathcal{T}_p)$, $\ell_{\mathcal{T}_c}(\Gamma) = \ell_{\mathcal{T}_p}(\Gamma)$ for every $\Gamma \in V(\mathcal{T}_c) - \{0, 00, 01\}$,
$\ell_{\mathcal{T}_c}(0) = \big(\ell_{\mathcal{T}_p}(0), \max\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\}\big)$,
$\ell_{\mathcal{T}_c}(\Gamma) = \big(\ell^1_{\mathcal{T}_p}(\Gamma), \bot\big)$ for every $\Gamma \in \{00, 11\}$,

(circ$_3$) the premise $\mathcal{T}_p$ is such that $V(\mathcal{T}_p) = \{\Lambda, 0, 1, 00, 01, 10, 11\}$,
$E(\mathcal{T}_p) = \big\{(\Gamma, \Gamma i) \mid \{\Gamma, \Gamma i\} \subseteq V(\mathcal{T}_p) \text{ and } i \in \{0,1\}\big\}$,
$\ell^1_{\mathcal{T}_p}(\Gamma i) = i$ for all $\Gamma \in \{\Lambda, 0, 1\}$ and $i \in \{0,1\}$,
$\ell^2_{\mathcal{T}_p}(\Gamma) = \bot$ for every $\Gamma \in V(\mathcal{T}_p) - \{0,1\}$,
$\ell^1_{\mathcal{T}_p}(\Lambda) = \text{root}$, $\{\ell^2_{\mathcal{T}_p}(0), \ell^2_{\mathcal{T}_p}(1)\} \subseteq \{0,1\}$,
the conclusion $\mathcal{T}_c$ is such that $V(\mathcal{T}_c) = V(T_p)$, $E(T_c) = E(\mathcal{T}_p)$,
$\ell^2_{\mathcal{T}_c}(\Gamma) = \ell^2_{\mathcal{T}_p}(\Gamma)$ for every $\Gamma \in V(\mathcal{T}_c) - \{\Lambda, 0, 1\}$,
$\ell_{\mathcal{T}_c}(\Gamma) = (\ell^1_{\mathcal{T}_p}, \bot)$ for every $\Gamma \in \{0,1\}$,
and $\ell_{\mathcal{T}_c}(\Lambda) = \big(\text{root}, \max\{\ell_{\mathcal{T}_p}(0), \ell_{\mathcal{T}_p}(1)\}\big)$.

The identity rules $\mathcal{T} \vdash \mathcal{T}$ are also necessary which are defined in a similar way as in Example 1.

There is no other rewriting rule of $\mathcal{M}_{\text{circ}}$ than that described by the above schemes.

The following lemma characterizes the computations of G–P–R machine $\mathcal{M}_{\text{circ}}$.

Lemma 3. *Let $\mathcal{T} \in \mathcal{S}_{\text{circ}}$ be a regular labelled binary tree such that the following conditions hold*

(a) *for every binary string Γ of length equal to the depth of $\mathcal{T}$ there exists a natural number i with $i > 0$ such that $\ell^2_{\mathcal{T}}(\Gamma \restriction i) \neq \bot$,*
(b) *for every $\Gamma \in V(\mathcal{T})$ and $j \in \{0,1\}$ if $\Gamma j \in V(\mathcal{T})$ and $\ell^2_{\mathcal{T}}(\Gamma j) \neq \bot$, then $\ell^2_{\mathcal{T}}(\Gamma\neg(j)) \neq \bot$, where $\neg(0) = 1$ and $\neg(1) = 0$.*

Then for

$$n = \max\big\{i \mid i \text{ is the length of some binary string } \Gamma \in V(\mathcal{T}) \text{ with } \ell^2_{\mathcal{T}}(\Gamma) \neq \bot\big\}$$

the value $\mathcal{F}^n_{\text{circ}}(\mathcal{T})$ is that regular labelled tree $\mathcal{T}'$ which is such that $V(\mathcal{T}') = V(\mathcal{T})$, $E(\mathcal{T}') = E(\mathcal{T})$, $\ell^2_{\mathcal{T}'}(\Gamma) = \bot$ for all $\Gamma \in V(\mathcal{T}') - \{\Lambda\}$ and $\ell^2_{\mathcal{T}'}(\Lambda) = \max\big\{\ell^2_{\mathcal{T}}(\Gamma) \mid \Gamma \in V(\mathcal{T}') \text{ and } \ell^2_{\mathcal{T}}(\Gamma) \neq \bot\big\}$, where $\mathcal{F}^n_{\text{circ}}(\mathcal{T})$ is defined inductively by $\mathcal{F}^1_{\text{circ}}(\mathcal{T}) = \mathcal{F}_{\text{circ}}(\mathcal{T})$ and $\mathcal{F}^n_{\text{circ}}(\mathcal{T}) = \mathcal{F}_{\text{circ}}(\mathcal{F}^{n-1}_{\text{circ}}(\mathcal{T}))$.

Example 3 (A G–P–R machine solving 3-SAT problem in a polynomial time). We use a Turing machine $\dot{\mathbb{T}}$ such that for every formula φ in a disjunctive normal form as in 3-SAT problem and every truth assignment T for variables of φ the machine decides in the time $\leq n^{k_0}$ whether φ is valid for T, where the ordered pair (φ, T) is an input for $\dot{\mathbb{T}}$ from which the machine begins the computation, k_0 is some constant natural number, and n is the number of variables occurring in φ. We claim for $\dot{\mathbb{T}}$ that:

(A) if n is the number of variables occurring in φ, then any truth assignment T for variables of φ is represented by that binary string Γ of length n in the machine tape which is such that if the value "True" is assigned to the i-th variable of φ, then 1 is the i-th element of Γ, otherwise the i-th element of Γ is 0,

(B) for the G–P–R machine $\mathcal{M}_{\dot{\mathbb{T}}}$ simulating the computations of $\dot{\mathbb{T}}$ if we have that

(b_1) $\mathcal{G}_{\varphi,\Gamma}$ is that instantaneous description of $\mathcal{M}_{\dot{\mathbb{T}}}$ which coincides with the initial instantaneous description or initial configuration for input (φ, T) with T represented by the binary string Γ in the machine tape as in (A),

(b_2) $\mathcal{G} = \mathcal{F}^q_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma})$ is the final instantaneous description of $\mathcal{M}_{\dot{\mathbb{T}}}$ for the case of the final or halting state "stop" reached by $\dot{\mathbb{T}}$ after q steps of computation starting with input (φ, T) with T related to Γ as in (A),

then $\mathcal{G}$ is a labelled graph induced by some acceptable triple (k, m, n) of integers with $(-k, m) \in E(\mathcal{G})$ such that $\ell_{\mathcal{G}}(0)$ is the final state "stop" and $\ell_{\mathcal{G}}(m) = 1$ if φ is valid for the truth assignment represented by Γ, otherwise $\ell_{\mathcal{G}}(m) = 0$, where $\mathcal{F}_{\dot{\mathbb{T}}}$ is the transition function of $\mathcal{M}_{\dot{\mathbb{T}}}$ and $\mathcal{F}^q_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma})$ is inductively defined: $\mathcal{F}^1_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma}) = \mathcal{F}_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma})$ and $\mathcal{F}^q_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma}) = \mathcal{F}_{\dot{\mathbb{T}}}(\mathcal{F}^{q-1}_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma}))$.

The shape of formulas in a disjunctive normal form in 3-SAT problem (it suffices to consider formulas of $n > 3$ variables which are disjunctions of $2^3 \cdot \binom{n}{3}$ nonrepetitive clauses, each conjunction of three literals containing different variables) provides that the claimed Turing machine $\dot{\mathbb{T}}$ can be constructed from some simpler three-string or three-tape Turing machine 3-$\mathbb{T}$ according to the general construction in the proof of Theorem 2.1, p. 30 of [4]. The first tape of 3-$\mathbb{T}$ is an input tape containing some presentation of a formula, the second tape is also an input tape containing some presentation of a truth assignment, and the third tape is an output tape. The machine 3-$\mathbb{T}$ reads only its input tapes and does not move its head or cursor on output tape printing or erasing the digits 0, 1. The machine 3-$\mathbb{T}$ reaches the final state in the time not greater than $2^3 \cdot n^5$ steps for a formula of n variables, hence by Theorem 2.1, p. 30 of [4] the machine $\dot{\mathbb{T}}$ reaches the final state in the time not greater than $2^6 \cdot n^{10}$ steps for a formula of n variables.

We outline a construction of a G–P–R machine $\mathcal{M}_{\text{3-SAT}}$ aimed to solve 3-SAT problem in a polynomial time. The initial instantaneous descriptions of $\mathcal{M}_{\text{3-SAT}}$ are labelled directed graphs $\mathcal{G}^0_\varphi$ determined by formulas φ in disjunctive normal forms as in 3-SAT problem in the following way:

(I_V) $V(\mathcal{G}^0_\varphi) = \big\{(j, \Gamma) \mid j \in V(\mathcal{G}_{\varphi,\Gamma}) \text{ and } \Gamma \in 2^n\big\} \cup \big\{(\Theta, \Lambda) \mid \Theta \in V(\mathcal{T}_\perp)\big\}$,
where $\mathcal{G}_{\varphi,\Gamma}$ is that initial instantaneous description which was introduced in (b_1), n is the number of variables occurring in φ, 2^n is the set of binary strings of length n, and $\mathcal{T}_\perp$ is the regular labelled binary tree of depth $n-1$ such that $\ell^2_{\mathcal{T}_\perp}(\Theta) = \perp$ for every $\Theta \in V(\mathcal{T}_\perp)$,

(I_E) $E(\mathcal{G}^0_\varphi) = \big\{\big((j, \Gamma), (j', \Gamma)\big) \mid (j, j') \in E(\mathcal{G}_{\varphi,\Gamma}) \text{ and } \Gamma \in 2^n\big\}$
$\cup \big\{\big((\Theta, \Lambda), (\Theta', \Lambda)\big) \mid (\Theta, \Theta') \in E(\mathcal{T}_\perp)\big\}$

$\cup\big\{\big((\Theta,\Lambda),(j_{\Theta i},\Theta i)\big) \mid \Theta i \in 2^n,\ i \in \{0,1\},\ \text{and}\ \Theta \in 2^{n-1}\big\}$,
where $j_{\Theta i}$ is that unique vertex of $\mathcal{G}_{\varphi,\Theta i}$ for which $\ell_{\mathcal{G}_{\varphi,\Theta i}}(j_{\Theta i}) = \%$,

(I$_\ell$) the labelling function $\ell_{\mathcal{G}^0_\varphi}$ is defined in the following way:

- $\ell_{\mathcal{G}^0_\varphi}((j,\Gamma)) = \ell_{\mathcal{G}_{\varphi,\Gamma}}(j)$ for all $\Gamma \in 2^n$ and $j \in V(\mathcal{G}_{\varphi,\Gamma})$ except j_Γ for which $\ell_{\mathcal{G}_{\varphi,\Gamma}}(j_\Gamma) = \%$,
- $\ell_{\mathcal{G}^0_\varphi}((j_\Gamma,\Gamma)) = (i,\%)$ if Γ is Θi for $i \in \{0,1\}$, where j_Γ is such that $\ell_{\mathcal{G}_{\varphi,\Gamma}}(j_\Gamma) = \%$,
- $\ell_{\mathcal{G}^0_\varphi}((\Theta,\Lambda)) = \ell_{\mathcal{T}_\perp}(\Theta)$ for every $\Theta \in V(\mathcal{T}_\perp)$.

Then we define inductively the labelled graphs $\mathcal{G}^k_\varphi$ for a natural number $k > 0$ and a formula φ in a disjunctive normal form as in 3-SAT problem such that the sets $V(\mathcal{G}^k_\varphi)$ and $E(\mathcal{G}^k_\varphi)$ are defined in an analogous way as $V(\mathcal{G}^0_\varphi)$ and $E(\mathcal{G}^0_\varphi)$ were defined in (I$_V$) and (I$_E$), respectively, except the graphs $\mathcal{G}_{\varphi,\Gamma}$ are replaced by $\mathcal{F}^k_{\dagger\mathbb{T}}(\mathcal{G}_{\varphi,\Gamma})$ (see the definition of $\mathcal{F}^q_{\dagger\mathbb{T}}(\mathcal{G}_{\varphi,\Gamma})$ in (B)). The labelling function $\ell_{\mathcal{G}^k_\varphi}$ of $\mathcal{G}^k_\varphi$ is determined by the labelling function of $\mathcal{G}^{k-1}_\varphi$ by imposing that $\mathcal{G}^k_\varphi$ is the result of simultaneous application to $\mathcal{G}^{k-1}_\varphi$ in G–P–R machine mode the rules of the G–P–R machines $\mathcal{M}_{\dagger\mathbb{T}}$ and $\mathcal{M}_{\text{circ}}$ with the label % replaced by $(i,\%)$ for $i \in \{0,1\}$, and the following new rules given by the scheme $\mathcal{G}_p \vdash \mathcal{G}_c$, where the premise $\mathcal{G}_p$ is such that

$V(\mathcal{G}_p) = \{\Lambda, 0, 00, 01, 001, 011, 0010, 0011, 0110, 0111\}$,
$E(\mathcal{G}_p) = \big\{(\Gamma,\Gamma i) \mid \{\Gamma,\Gamma i\} \subseteq V(\mathcal{G}_p) - \{0010, 0110\} \text{ and } i \in \{0,1\}\big\}$
$\quad\cup\{(0010,001),(0110,011),(0,0)\}$,
$\ell^2_{\mathcal{G}_p}(\Lambda) = \ell^2_{\mathcal{G}_p}(0) = \perp$ [2], $\{\ell^1_{\mathcal{G}_p}(\Lambda), \ell^1_{\mathcal{G}_p}(0)\} \subseteq \{0,1\}$,
$\ell_{\mathcal{G}_p}(00) = (0,\%)$, $\ell_{\mathcal{G}_p}(01) = (1,\%)$, $\{\ell_{\mathcal{G}_p}(001), \ell_{\mathcal{G}_p}(011)\} \subseteq \{0,1\}$,
$\ell_{\mathcal{G}_p}(0010) = \ell_{\mathcal{G}_p}(0110) = \text{“stop”} \in Q$, $\{\ell_{\mathcal{G}_p}(0011), \ell_{\mathcal{G}_p}(0111)\} \subseteq \Sigma$,

the conclusion $\mathcal{G}_c$ is such that $V(\mathcal{G}_c) = V(\mathcal{G}_p)$, $E(\mathcal{G}_c) = E(\mathcal{G}_p)$,
$\ell_{\mathcal{G}_c}(v) = \ell_{\mathcal{G}_p}(v)$ for every $v \in V(\mathcal{G}_p)$ except $\ell_{\mathcal{G}_c}(001) = \ell_{\mathcal{G}_c}(011) = \perp$,
and $\ell^2_{\mathcal{G}_c}(0) = \max\{\ell_{\mathcal{G}_p}(001), \ell_{\mathcal{G}_p}(011)\}$, $\ell^1_{\mathcal{G}_c}(0) = \ell^1_{\mathcal{G}_p}(0)$.

Thus we define the set $\mathcal{S}_{\text{3-SAT}}$ of instantaneous descriptions of G–P–R machine $\mathcal{M}_{\text{3-SAT}}$ by

$$\mathcal{S}_{\text{3-SAT}} = \big\{\mathcal{G}^k_\varphi \mid k \text{ is a natural number and } \varphi \text{ is a formula in a disjunctive normal form as in 3-SAT problem}\big\}.$$

The transition function $\mathcal{F}_{\text{3-SAT}}$ of G–P–R machine $\mathcal{M}_{\text{3-SAT}}$ is given by

$$\mathcal{F}_{\text{3-SAT}}(\mathcal{G}^k_\varphi) = \mathcal{G}^{k+1}_\varphi \text{ for every } k \geq 0 \text{ and every } \varphi.$$

The rewriting rules of $\mathcal{M}_{\text{3-SAT}}$ are the rewriting rules of the G–P–R machines $\mathcal{M}_{\dagger\mathbb{T}}$, $\mathcal{M}_{\text{circ}}$ with the label % replaced by $(i,\%)$ for $i \in \{0,1\}$, and the new rules introduced above.

Theorem 2. *The* G–P–R *machine* $\mathcal{M}_{\text{3-SAT}}$ *solves* 3-SAT *problem in a polynomial time.*

[2] We assume that $\perp \notin \Sigma \cup Q \cup \{\%, \S\}$.

Proof. Since the upper bound of the time of computation of $\dot{\mathbb{T}}$ does not depend on the binary sequences Γ representing truth assignments but only on their length and is polynomial with respect to this length, the upper bound of the time of computation of G–P–R machine $\mathcal{M}_{\dot{\mathbb{T}}}$ also does not depend on binary sequences Γ representing truth assignments and is polynomial with respect to the length of these sequences Γ. Hence by Lemma 3 we get the theorem.

Less formally, for a given formula φ of n variables with $n > 3$ the machine $\mathcal{M}_{\text{3-SAT}}$ simultaneously simulates without any delay the computations of 2^n copies of the Turing machine $\dot{\mathbb{T}}$, where 2^n possible truth assignments for φ are the inputs together with φ itself for these 2^n copies of $\dot{\mathbb{T}}$, respectively. Here each truth assignment T is associated to that copy of $\dot{\mathbb{T}}$ which is aimed to decide whether φ is valid for T.

Then Boolean circuit part of $\mathcal{M}_{\text{3-SAT}}$ simulates the computation of tree-like Boolean circuit $\mathcal{C}$ of 2^n input gates, where the underlying graph of $\mathcal{C}$ is a tree of depth n and all non-input gates of $\mathcal{C}$ are OR gates. The 2^n input gates of $\mathcal{C}$ receive those inputs which are the output results of the computations of the above 2^n copies of $\dot{\mathbb{T}}$, respectively. Here each input gate g is associated with that copy $\dot{\mathbb{T}}_g$ of $\dot{\mathbb{T}}$ for which g is connected with that unique vertex i of the final graphical instantaneous description of $\dot{\mathbb{T}}_g$ for which $(0, i)$ is an edge of this final graphical instantaneous description and i is labelled by the output result of $\dot{\mathbb{T}}_g$ with 0 labelled by the final or halting state of $\dot{\mathbb{T}}_g$. The inputs of $\mathcal{C}$ are simultaneously processed by $\mathcal{C}$ to give the output result in the root of the underlying graph of $\mathcal{C}$. The output result contained in the root yields an answer to a question whether there exists a truth assignment for φ such that φ is valid for this assignment. Therefore $\mathcal{M}_{\text{3-SAT}}$ solves 3-SAT problem in a polynomial time. □

Corollary. *There exists a Gandy machine which solves* 3-SAT *problem in a polynomial time but with the exponential number of urelement processors.*

Proof. The corollary is a consequence of Theorems 1 and 2. □

4 Concluding remarks

One could adopt G–P–R machines and Gandy machines as the underlying abstract computing devices of computational complexity theory because these machines propose a wide scope of possible computational parallelism, even up to unreliable parallelism of G–P–R machine $\mathcal{M}_{\text{3-SAT}}$ and representing it Gandy machine which prove that polynomial computational time does not imply polynomial computational space understood as the size of hardware measured by the number of urelement (indecomposable) processors of a machine. We will show in a forthcoming paper about randomized G–P–R machines those G–P–R machines which are capable to construct in polynomial time the initial instantaneous descriptions of the machine $\mathcal{M}_{\text{3-SAT}}$ from simpler labelled graphs of size, i.e., the number of vertices, depending linearly on the size of input data of a formula and a truth assignment, where some versions of division rules of membrane computing [6] are used.

Appendix. Graph-theoretical and category-theoretical preliminaries

A [*finite*] *labelled directed graph over* a set Σ of labels is defined to be an ordered triple $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}), \ell_{\mathcal{G}})$, where $V(\mathcal{G})$ is a [finite] *set of vertices* of $\mathcal{G}$, $E(\mathcal{G})$ is a subset of $V(\mathcal{G}) \times V(\mathcal{G})$ called the *set of edges* of $\mathcal{G}$, and $\ell_{\mathcal{G}}$ is a function from $V(\mathcal{G})$ into Σ called the *labelling function* of $\mathcal{G}$. We drop the adjective 'directed' if there is no risk of confusion.

A *homomorphism of a labelled directed graph* $\mathcal{G}$ *over* Σ *into a labelled directed graph* $\mathcal{G}'$ *over* Σ is an ordered triple $(\mathcal{G}, \mathrm{h} : V(\mathcal{G}) \to V(\mathcal{G}'), \mathcal{G}')$ such that h is a function from $V(\mathcal{G})$ into $V(\mathcal{G}')$ which satisfies the following conditions:

(H_1) $(v, v') \in E(\mathcal{G})$ implies $(\mathrm{h}(v), \mathrm{h}(v')) \in E(\mathcal{G}')$ for all $v, v' \in V(\mathcal{G})$,
(H_2) $\ell_{\mathcal{G}'}(\mathrm{h}(v)) = \ell_{\mathcal{G}}(v)$ for every $v \in V(\mathcal{G})$.

If a triple $h = (\mathcal{G}, \mathrm{h} : V(\mathcal{G}) \to V(\mathcal{G}'), \mathcal{G}')$ is a homomorphism of a labelled directed graph $\mathcal{G}$ over Σ into a labelled directed graph $\mathcal{G}'$ over Σ, we denote this triple by $h : \mathcal{G} \to \mathcal{G}'$, we write $\mathrm{dom}(h)$ and $\mathrm{cod}(h)$ for $\mathcal{G}$ and $\mathcal{G}'$, respectively, according to category theory convention, and we write $h(v)$ for the value $\mathrm{h}(v)$.

A homomorphism $h : \mathcal{G} \to \mathcal{G}'$ of labelled directed graphs over Σ is an *embedding of* $\mathcal{G}$ *into* $\mathcal{G}'$, denoted by $h : \mathcal{G} \rightarrowtail \mathcal{G}'$, if the following condition holds:

(E) $h(v) = h(v')$ implies $v = v'$ for all $v, v' \in V(\mathcal{G})$.

An embedding $h : \mathcal{G} \rightarrowtail \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ is an *inclusion of* $\mathcal{G}$ *into* $\mathcal{G}'$, denoted by $h : \mathcal{G} \hookrightarrow \mathcal{G}'$, if the following holds:

(I) $h(v) = v$ for every $v \in V(\mathcal{G})$.

We say that a labelled directed graph $\mathcal{G}$ over Σ is a *labelled subgraph* of a labelled directed graph $\mathcal{G}'$ over Σ if there exists an inclusion $h : \mathcal{G} \hookrightarrow \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ.

For an embedding $h : \mathcal{G} \rightarrowtail \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ we define the *image* of h, denoted by $\mathrm{im}(h)$, to be a labelled directed graph $\widehat{\mathcal{G}}$ over Σ such that $V(\widehat{\mathcal{G}}) = \{h(v) \mid v \in V(\mathcal{G})\}$, $E(\widehat{\mathcal{G}}) = \{(h(v), h(v')) \mid (v, v') \in E(\mathcal{G})\}$, and the labelling function $\ell_{\hat{\mathcal{G}}}$ of $\widehat{\mathcal{G}}$ is the restriction of the labelling function $\ell_{\mathcal{G}'}$ of $V(\mathcal{G}')$ to the set $V(\widehat{\mathcal{G}})$, i.e., $\ell_{\hat{\mathcal{G}}}(v) = \ell_{\mathcal{G}'}(v)$ for every $v \in V(\widehat{\mathcal{G}})$.

A homomorphism $h : \mathcal{G} \to \mathcal{G}'$ of labelled directed graphs over Σ is an *isomorphism* of $\mathcal{G}$ into $\mathcal{G}'$ if there exists a homomorphism $h^{-1} : \mathcal{G}' \to \mathcal{G}$ of labelled directed graphs over Σ, called the inverse of h, such that the following conditons hold:

(Iz_1) $h^{-1}(h(v)) = v$ for every $v \in V(\mathcal{G})$, (Iz$_2$) $h(h^{-1}(v)) = v$ for every $v \in V(\mathcal{G}')$.

We say that a labelled directed graph $\mathcal{G}$ over Σ *is isomorphic* to a labelled directed graph $\mathcal{G}'$ over Σ if there exists an isomorphism $h : \mathcal{G} \to \mathcal{G}'$ of labelled graphs $\mathcal{G}, \mathcal{G}'$ over Σ.

For an embedding $h : \mathcal{G} \rightarrowtail \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ we define a homomorphism $\dot{h} : \mathcal{G} \to \mathrm{im}(h)$ by $\dot{h}(v) = h(v)$ for every $v \in V(\mathcal{G})$. This homomorphism $\dot{h}$ is an isomorphism of $\mathcal{G}$ into $\mathrm{im}(h)$, called an *isomorphism deduced by h*.

For a labelled directed graph $\mathcal{G}$ over Σ, the *identity homomorphism* (or simply, *identity of* $\mathcal{G}$), denoted by $\mathrm{id}_{\mathcal{G}}$, is the homomorphism $h : \mathcal{G} \to \mathcal{G}$ such that $h(v) = v$ for every $v \in V(\mathcal{G})$.

We say that a labelled directed graph $\mathcal{G}$ over Σ is an *isomorphically perfect* labelled directed graph over Σ if the identity homomorphism $\mathrm{id}_{\mathcal{G}}$ is a unique isomorphism of labelled directed graph $\mathcal{G}$ into $\mathcal{G}$.

Lemma 4. *Let $\mathcal{G}$ be an isomorphically perfect labelled directed graph over Σ and let $h : \mathcal{G} \to \mathcal{G}'$, $h' : \mathcal{G} \to \mathcal{G}'$ be two isomorphisms of labelled graphs $\mathcal{G}, \mathcal{G}'$ over Σ. Then $h = h'$.*

We say that a set or a class $\mathcal{A}$ of labelled directed graphs over Σ is *skeletal* if for all labelled directed graphs $\mathcal{G}, \mathcal{G}'$ in $\mathcal{A}$ if they are isomorphic, then $\mathcal{G} = \mathcal{G}'$.

A *gluing diagram* $\mathcal{D}$ of labelled directed graphs over Σ is defined by:

— its *set $\mathcal{I}$ of indexes* with a distinguished index $\Delta \in \mathcal{I}$, called the *center* of $\mathcal{D}$,
— its *family* $\mathcal{G}_i$ $(i \in \mathcal{I})$ *of labelled directed graphs* over Σ,
— its *family* gl_i $(i \in \mathcal{I} - \{\Delta\})$ *gluing conditions* which are sets of ordered pairs such that
 (i) $\mathrm{gl}_i \subseteq V(\mathcal{G}_\Delta) \times V(\mathcal{G}_i)$ for every $i \in \mathcal{I} - \{\Delta\}$,
 (ii) $(v, v') \in \mathrm{gl}_i$ implies $\ell_{\mathcal{G}_\Delta}(v) = \ell_{\mathcal{G}_i}(v)$ for all $v \in V(\mathcal{G}_\Delta)$, $v' \in V(\mathcal{G}_i)$, and for every $i \in \mathcal{I} - \{\Delta\}$,
 (iii) for every $i \in \mathcal{I} - \{\Delta\}$ if gl_i is non-empty, then there exists a bijection

$$b_i : L(\mathrm{gl}_i) \to R(\mathrm{gl}_i)$$

for $L(\mathrm{gl}_i) = \{v \mid (v, v') \in \mathrm{gl}_i \text{ for some } v'\}$ and $R(\mathrm{gl}_i) = \{v' \mid (v, v') \in \mathrm{gl}_i \text{ for some } v\}$ such that $\big\{(v, b_i(v)) \mid v \in L(\mathrm{gl}_i)\big\} = \mathrm{gl}_i$.

For a gluing diagram $\mathcal{D}$ of labelled directed graphs over Σ we define a *cocone* of $\mathcal{D}$ to be a family $h_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$ of homomorphisms of labelled directed graphs over Σ (here $\mathrm{cod}(h_i) = \mathcal{G}$ for every $i \in \mathcal{I}$) such that

$$l_{\mathcal{G}}(h_\Delta(v)) = l_{\mathcal{G}}(h_i(v'))$$

for every pair $(v, v') \in \mathrm{gl}_i$ and every $i \in \mathcal{I} - \{\Delta\}$.

A cocone $q_i : \mathcal{G}_i \to \widetilde{\mathcal{G}}$ $(i \in \mathcal{I})$ of $\mathcal{D}$ is called a *colimiting cocone of* $\mathcal{D}$ if for every cocone $h_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$ of $\mathcal{D}$ there exists a unique homomorphism

$h : \widetilde{\mathcal{G}} \to \mathcal{G}$ of labelled directed graphs $\widetilde{\mathcal{G}}, \mathcal{G}$ over Σ such that $h(q_i(v)) = h_i(v)$ for every $v \in V(\mathcal{G}_i)$ and for every $i \in \mathcal{I}$. The labelled directed graph $\widetilde{\mathcal{G}}$ is called a *colimit* of $\mathcal{D}$, the homomorphisms q_i $(i \in \mathcal{I})$ are called *canonical injections* and the unique homomorphism h is called the *mediating morphism* for $h_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$.

For a gluing diagram $\mathcal{D}$ one constructs its colimit $\widetilde{\mathcal{G}}$ in the following way:

— $V(\widetilde{\mathcal{G}}) = \bigcup\limits_{i \in \mathcal{I}} (V_i \times \{i\})$, where
$V_\Delta = V(\mathcal{G}_\Delta)$ for the center Δ of $\mathcal{D}$,
$V_i = V(\mathcal{G}_i) - R(\mathrm{gl}_i)$ for every $i \in \mathcal{I} - \{\Delta\}$,

— $E(\widetilde{\mathcal{G}}) = \bigcup\limits_{i \in \mathcal{I}} E_i$, where
$E_\Delta = \big\{\big((v,\Delta),(v',\Delta)\big) \mid (v,v') \in E(\mathcal{G}_\Delta)\big\}$ for the center Δ of $\mathcal{D}$,

$$\begin{aligned} E_i = {} & \big\{\big((v,i),(v',i)\big) \mid (v,v') \in E(\mathcal{G}_i) \text{ and } \{v,v'\} \subseteq V_i\big\} \\ & \cup \big\{\big((v,\Delta),(v',\Delta)\big) \mid (v,v'') \in \mathrm{gl}_i,\ (v',v''') \in \mathrm{gl}_i, \\ & \qquad \text{and } (v'',v''') \in E(\mathcal{G}_i) \text{ for some } v'',v'''\big\} \\ & \cup \big\{\big((v,\Delta),(v',i)\big) \mid v' \in V_i,\ (v,v'') \in \mathrm{gl}_i \text{ and } (v'',v') \in E(\mathcal{G}_i) \text{ for some } v''\big\} \\ & \cup \big\{\big((v,i),(v',\Delta)\big) \mid v \in V_i,\ (v,v'') \in \mathrm{gl}_i \text{ and } (v,v'') \in E(\mathcal{G}_i) \text{ for some } v''\big\} \end{aligned}$$

for every $i \in \mathcal{I} - \{\Delta\}$,

— the labelling function $\ell_{\widetilde{\mathcal{G}}}$ is defined by $\ell_{\widetilde{\mathcal{G}}}((v,i)) = \ell_{\mathcal{G}_i}(v)$ for every $(v,i) \in V(\widetilde{\mathcal{G}})$.

The definition of a colimiting cocone of a gluing diagram $\mathcal{D}$ provides that any other colimit of $\mathcal{D}$ is isomorphic to the colimit of $\mathcal{D}$ constructed above. Hence one proves the following lemma.

Lemma 5. *Let $\mathcal{D}$ be a gluing diagram of labelled graphs over Σ. Then for every colimiting cocone $q_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$ of $\mathcal{D}$ if $i' \neq i''$, then*

$$\big(V(\mathrm{im}(q_{i'})) - V(\mathrm{im}(q_\Delta))\big) \cap \big(V(\mathrm{im}(q_{i''})) - V(\mathrm{im}(q_\Delta))\big) = \varnothing$$

for all $i', i'' \in \mathcal{I} - \{\Delta\}$, where Δ is the center of $\mathcal{D}$ and the elements of nonempty $V(\mathrm{im}(q_i)) - V(\mathrm{im}(q_\Delta))$ with $i \neq \Delta$ are 'new' elements and the elements of $V(\mathrm{im}(q_\Delta))$ are 'old' elements.

References

1. R. Gandy, *Church's thesis and principles for mechanisms*, in: The Kleene Symposium, eds. J. Barwise et al., North-Holland, Amsterdam 1980, pp. 123–148.
2. *Handbook of graph grammars and computing by graph transformation.* Vol. 1. *Foundations*, ed. by G. Rozenberg, World Scientific, River Edge, NJ, 1997; Vol. 2. *Applications, languages and tools*, ed. by H. Ehrig et al., World Scientific, River Edge, NJ, 1999; Vol. 3. *Concurrency, parallelism, and distribution*, ed. by H. Ehrig et al., World Scientific, River Edge, NJ, 1999.

3. P. Hartmann, *Parallel replacement systems on geometric hypergraphs: a mathematical tool for handling dynamic geometric sceneries*, in: Parcella '94, Akademie Verlag, Potsdam 1994, pp. 81–90.
4. G. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, Mass. 1994.
5. Gh. Păun, *P systems with active membranes: Attacking NP complete problems*, Journal of Automata, Languages and Combinatorics 6 (2000), pp. 75–90.
6. Gh. Păun, *Membrane Computing. An Introduction*, Springer, Berlin 2002.
7. W. Sieg, J. Byrnes, *An abstract model for parallel computations*: *Gandy's Thesis*, The Monist 82:1 (1999), 150–164.
8. E. Steinhart, *Logically Possible Machines*, Mind and Machines 12 (2002), pp. 259–280.
9. J. Wiedermann, *Can Cognitive and Intelligent Systems Outperform Turing Machines?*, `http://www.cs.cas.cz/semweb/download.php?file=05-06-Wiedermann&type=pdf`

Linking Bistable Dynamics to Metabolic P Systems

Roberto Pagliarini[1], Luca Bianco[2], Vincenzo Manca[1], Conrad Bessant[2]

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

[2] Bioinformatics Group, Building 63, Cranfield University
Cranfield, Bedfordshire, MK43 0AL, UK
{l.bianco,c.bessant}@cranfield.ac.uk

Summary. Bistability, or more generally multistability, is an important recurring theme in biological systems. In particular, the discovery of bistability in signal pathways of genetic networks, prompts strong interest in understanding both the design and function of these networks. Therefore, modelling these systems is crucial to understand their behaviors, and also to analyze and identify characteristics that would otherwise be difficult to realize. Although different classes of models have been used to study bistable dynamics, there is a lag in the development of models for bistable systems starting from experimental data. This is due to the lack of detailed knowledge of biochemical reactions and kinetic rates.

In this work, we propose a procedure to develop, starting from observed dynamics, Metabolic P models for multistable processes. As a case study, a mathematical model of the Schlögel's dynamics, which represents an example of a chemical reaction system that exhibits bistability, is inferred starting from observed stochastic bistable dynamics. Since, recent experiments indicate that noise plays an important role in the switching of bistable systems, the success of this work suggests that this approach is a very promising one for studying dynamics and role of noise in biological systems, such as, for example, genetic regulatory networks.

1 Introduction

Bistability is an important recurring theme in cell signaling and has been studied extensively through experiments, theoretical analysis, and numerical simulations [8, 10, 3, 22]. A bistable system has two distinct steady states and any initial state will eventually bring the system into one of them. Bistability is a key to understand basic cellular phenomena, such as decision-making processes in cell cycle progression, cell differentiation and apoptosis [2]. It is also involved in the loss of cellular homeostasis associated with early events in cancer onset [12] and in prior diseases [11]. In [23], different bistable phenomena in bacteria and the importance of bistability for the origin of new species are studied.

Recently, interest grew in the investigation of the bistable dynamics regulations through differential and stochastic modelling [8, 10]. Usually, differential and stochastic models are developed based on detailed knowledge of biochemical reactions, molecule amounts and kinetic. Kinetic rates are estimated by using the mass action law, while stochastic parameters are derived from these rates.

However, there are several limitations for a reliable application of these classes of models. First, the determination of kinetic and stochastic constants depends on the chemo-physical details of the reactions, and moreover, even if carefully established in rigorous experimental settings, they may be completely different when many reactions are put together in real complex systems. Second, data availability and regulatory information usually can not provide a comprehensive picture of biological regulations. Lastly, in the processes with only a few molecules, classic mass action kinetics are no longer valid for describing the reaction dynamics. For these reasons, the modelling of observed dynamics is not a trivial work and in some cases it still remains an open problem.

Recognizing the importance of bistability in biochemical systems, some techniques to obtain mathematical models of bistable (multistable) systems starting from observed dynamics or experimental data are needed.

In this work, we propose a procedure, rooted in Metabolic P Systems [17, 15], shortly MP Systems, to infer models of an observed, also stochastic, bistable (multistable) dynamics.

As a case study, a mathematical model of the Schlögel's reaction [21], which represents an example of chemical reaction system that exhibits bistability, is inferred starting from observed stochastic dynamics. Since recent experiments indicate that noise and stochasticity play important roles in the switching of bistable systems, this work suggests that this approach is very promising for studying the dynamics and role of noise in biological systems, like, for example, genetic regulatory networks.

2 Metabolic P Systems and Log-Gain theory: a brief introduction

MP systems have been introduced as mono-membrane multiset rewriting grammars, whose rules are regulated by specific functions [15]. The aim is to control the matter transformation in a reactor by means of rules whose fluxes dynamically depend on the state of the system. This strategy of rules application is different to that of P Systems [20] and it has been successfully applied to several biological processes [1, 18, 19].

Specifically, an MP system is completely specified (the reader can find the formal definition in [17]) by: *i)* n substances and their initial values, *ii)* m reactions, with[3] $m > n$, *iii)* m corresponding flux regulation functions, *iv)* k parameters[4], and their initial values, which are arguments, beside substances, of flux regulation functions, and *v)* k parameter evolution functions.

A *state* q is an $\mathbb{R}^{n+k}$ vector, reporting the current amounts of substances and parameters, while each rule r_j (with $j = 1, \dots, m$) having some of the n substances as substrates and some as products, is associated with a couple of vectors $(r_j^-, r_j^+) \in \mathbb{N}^n \times \mathbb{N}^n$ (one of which possibly null), reporting the substance quantities respectively occurring in the premise and in the consequence of r_j.

As an example, we can consider a system $\mathcal{M}$ with three substances $\{a, b, c\}$, two parameters $\{v, w\}$ which values evolve, for $t \in \mathbb{N}$, according with their own function $f_v(t)$ and $f_w(t)$, respectively, and four reactions:

$$\begin{aligned} r_1 &: ab \rightarrow aa \\ r_2 &: bcc \rightarrow a \\ r_3 &: ac \rightarrow \lambda \\ r_4 &: abc \rightarrow bb. \end{aligned} \tag{1}$$

The reactions (1) correspond to the following vectors, respectively :

$$\begin{aligned} (r_1^-, r_1^+) = ((1,1,0),(2,0,0)),\ (r_2^-, r_2^+) = ((0,1,2),(1,0,0)), \\ (r_3^-, r_3^+) = ((1,0,1),(0,0,0)),\ (r_4^-, r_4^+) = ((1,1,1),(0,2,0)). \end{aligned}$$

Four flux regulation functions, one for each rule, are defined from $\mathbb{R}^5$ to $\mathbb{R}$, and they produce at each step *fluxes* u_1, u_2, u_3, u_4, associated with the corresponding reaction.

We call *stoichiometric matrix*, the $(n \times m)$-dimensional matrix $\mathbb{A}$ formed by the vectors $r_i^+ - r_i^-$, for every rule r_i, disposed according to a prefixed order. For example, in the system above, we have

$$\mathbb{A} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ 0 & -2 & -1 & -1 \end{pmatrix}. \tag{2}$$

The stoichiometric matrix is assumed to have maximal rank. Should we have one row linearly dependent on the others, we could delete it (together with the corresponding substance in the system, as studying its dynamics would not add any useful information on the system), and analyse only the remaining substances (we newly say n) and the corresponding $n \times m$ stoichiometric matrix which now has full rank.

[3] We assume $m > n$, as it realistically happens in biochemical systems. A few examples are given by the following protein-protein interaction networks: yeast has 8868 known interactions among 3280 proteins [9], Drosophila has 4780 known interactions among 4679 proteins, and C. elegans has 5534 known interactions among 3024 proteins [7].

[4] Parameters are internal or external controlling variables which somehow affect the system's functioning.

Let $U[t] = (u_r[t] \mid r \in R)$ be, for $t = 0, 1, \ldots$, the (m)-dimensional column vector of fluxes and $X[t]$ be the (n)-dimensional column vector of substances. Then, the dynamics of an MP system, given by both the evolution of parameters, according to their laws, and by the evolution of the substances, are computed by the *Equational Metabolic Algorithm*[15, 14], which is the following recurrent n-equations system:

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

where $\times$ denotes the ordinary matrix product and t the discrete instant of time.

This way to observe the evolution rules of a system reproducing a biological reaction has been proposed in [1] and constitutes a new perspective. In fact, by using MP systems, one assumes an a priori choice of the time interval τ, between consecutive evolution steps, that depends on the macroscopic level at which considering the dynamics of the system. Then, the flux values, depending on the state of the system, are computed according to the chosen observation granularity.

Therefore, the approach of modelling by MP systems considers the rules as macroscopic matter transformation reactions rather than microscopic molecular interactions. Then, the search of fluxes is aimed at designing a model of the observed macroscopic reality with respect to the abstract transformations one has assumed, and it is different from the rate estimation typically studied in systems biology.

This inverse dynamical problem is the starting point of the Log-Gain theory [16]. The goal of this theory is to deduce the time-series of fluxes, reproducing an observed dynamics and biologically meaningful, starting from some consecutive (at a time interval τ) time-series of the state of a system. When such time-series are known, the discovery of flux regulation functions is a problem of approximation which can be solved with mathematical regression techniques.

According to the simplest formulation of this theory, given a number of observations of the system's states, for which the stoichiometry is known, the relative variations of any reaction flux of the rule $r_j : \alpha_j \to \beta_j$ is the sum of the relative variations of its reactants, plus some error p_j, called *reaction offset*, which is introduced as a variable of the system:

$$(u_j[t+1] - u_j[t])/(u_j[t]) = \sum_{x \in \alpha_j} ((x[t+1] - x[t])/x[t]) + p_j.$$

We denote with $P[t]$ the m-dimensional vector of p_j variables, $j = 1, \ldots, m$, that is, of the errors introduced with the log-gain approximations of fluxes at step t. Furthermore, we denote with $Lg(U[t])$ the m-dimensional vector of relative fluctuations, that is $((u_j[t+1] - u_j[t])/u_j[t] \mid j = 1, \ldots, m)$, for any $t \in \mathbb{N}$. Analogously, $Lg(X[t])$ is the vector of relative variations of substances. Therefore, in formal terms, the $m+n$ equations system we want to solve (in order to find the vector U[t+1]) is

$$\begin{cases} Lg(U[t]) = \mathbb{B} \times Lg(X[t]) + C \cdot P[t+1] \\ \mathbb{A} \times U[t+1] = X[t+2] - X[t+1] \end{cases} \tag{4}$$

where $\mathbb{B}$ is a $(m \times n)$-dimensional boolean matrix selecting, by matrix product, the reactants for each reaction, and C is an m-dimensional boolean vector selecting, by Schur product, only n of the m reaction offsets (hence that are n other unknowns in the system, besides the m fluxes). As proposed in [4], vector C selects a set of n linear independent reactions, called *covering set*, in order to infer the flux time-series. However, a way to choose the best covering set among the linearly independent ones still remains to be found.

3 A flowchart to infer bistable MP systems

In order to model bistable (stochastic) phenomena starting from experimental data, we propose the flowchart represented in Figure 1. In our method, first a set of intermediate MP systems having dynamics in accordance with several observed dynamics is created, by using Log-Gain theory, least-square theory and correlation analysis. In the case of stochastic phenomena, this phase in particularly important because it allows us to obtain dynamics having behaviours, in particular those related to the reaction fluxes, not affected by observed noise. Namely, first *i)* we apply the Log-Gain theory to infer the numeric values of the reaction fluxes, then *ii)* a correlation analysis is used to suggest relationships between flux and substance/parameter time-series, and finally *iii)* least-squares theory is used to approximate the flux regulation functions.

Once we obtained good approximations of the different observed dynamics, we apply again the least-squares theory to infer an unique MP system modelling the bistability of the input phenomenon.

This last phase represents the major challenge because it needs a mathematical analysis to identify the appropriate forms of the final flux regulation functions. As result of this flowchart, we will obtain an MP system $\mathcal{M}$ modelling the bistability of the studied phenomenon.

In the following section we will apply this flowchart to a chemical reaction system which exhibits a stochastic bistable dynamics.

4 A case study: the stochastic Schlögel's reaction

An interesting example of bistable process is provided by the Schlögel's model [21], which is an autocatalytic, trimolecular reaction schema composed of the set of coupled chemical reactions reported in Table 1.

What makes the Schlögel's reaction especially interesting is that, despite its simplicity (it is composed of four reactions involving three species, two of which are buffered) it provides a very rich dynamics. If we fix the stochastic parameters (and hence the reaction rates which can be computed from them), according to certain ranges of values, as well as the initial amounts of the two buffered species a and b, depending on the initial amount of x, the stochastic simulation of the system provides a bistable behaviour.

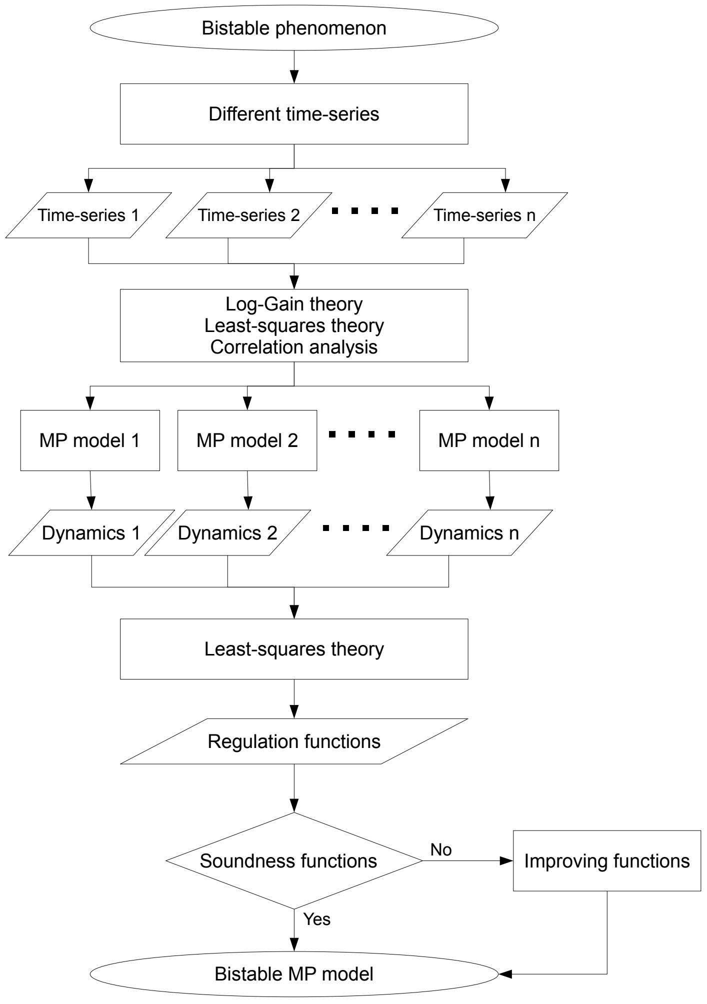


Fig. 1. Flowchart for the estimation of an MP system describing the dynamics of a bistable (stochastic) phenomenon. Experimental data are analysed and used to infer intermediated MP models which characterize the different observed dynamics. Then, these models are used to generate reaction flux and substance dynamics in accordance with the observed ones. Finally, these dynamics represent the input of a least-squares analysis, where, by using also some hypothesis about the logic governing the studied phenomenon, the final MP system is inferred.

Let $\#X[t] = (\#x_1[t], \#x_2[t], \ldots, \#x_n[t])$ be the vector representing the state of the system (i.e. the number of molecules of every species x_i in the system

Reactions	Stochastic parameters
$r_1 : a + 2x \longrightarrow 3x + a$	$c_1 = 3 \cdot 10^{-7}$
$r_2 : 3x \longrightarrow 2x$	$c_2 = 1 \cdot 10^{-4}$
$r_3 : b \longrightarrow x + b$	$c_3 = 1 \cdot 10^{-3}$
$r_4 : x \longrightarrow \lambda$	$c_4 = 3.5$

Table 1. Schlögel's reactions and a set of stochastic parameters [13].

evaluated at time t, for $i = 1, ..., n$). In our simulations, we used the set of stochastic parameters given in Table 1 and we performed, by using the *Stochastic Simulation Algorithm* [5, 6], shortly *SSA*, up to 40 independent simulations from $t = 0$ starting from the following initial configurations:

$$\begin{aligned}
\#X[0] &= (\#a[0], \#b[0], \#x[0]) = (1 \cdot 10^5, 2 \cdot 10^5, 0) \\
\#X[0] &= (\#a[0], \#b[0], \#x[0]) = (1 \cdot 10^5, 2 \cdot 10^5, 238) \\
\#X[0] &= (\#a[0], \#b[0], \#x[0]) = (1 \cdot 10^5, 2 \cdot 10^5, 800).
\end{aligned} \tag{5}$$

The average behaviours (i.e. simulations were sampled with constant rate and an average concentration of species x over the up to 40 different stochastic simulations starting from the same initial state was computed) of the species x, according to different initial states, are reported in Figures 2 and 3. The left-hand part of Figure 2 shows that if we simulate the Schlögel's model starting from the initial state $(1 \cdot 10^5, 2 \cdot 10^5, 0)$ then the number of molecules of species x goes up until it reaches a stable 'on' state. Alike, the right-hand part of the same Figure shows that if we simulate it starting from the initial state $(1 \cdot 10^5, 2 \cdot 10^5, 800)$ then the number of molecules of species x decreases and stabilizes at stable 'off' state. Moreover, if we start the simulations from the initial state $(1 \cdot 10^5, 2 \cdot 10^5, 238)$, the number of molecules of x randomly stabilizes at one of the two possible distinct stable states. These behaviours are depicted in Figure 3.

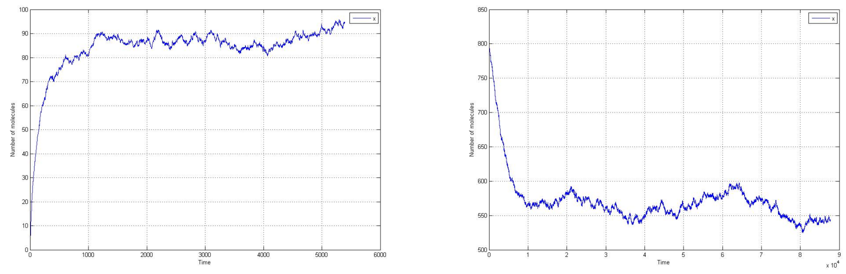


Fig. 2. Average time-evolution of species x in the Schlögel model obtained by using the SSA and considering the initial states $(1 \cdot 10^5, 2 \cdot 10^5, 0)$ (leftmost image) and $(1 \cdot 10^5, 2 \cdot 10^5, 800)$ (rightmost image).

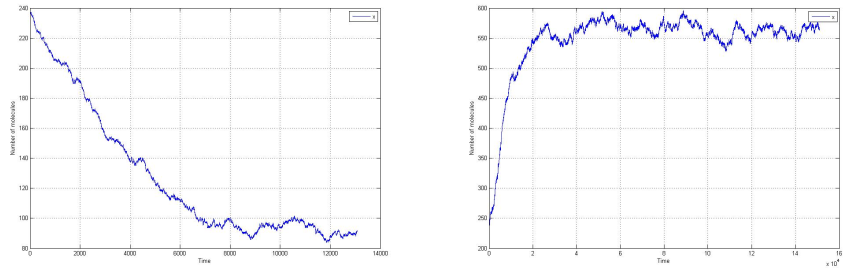


Fig. 3. Average time-evolution of species x in the Schlögel by using the SSA with initial state $(1 \cdot 10^5, 2 \cdot 10^5, 238)$. The two different behaviours are the result of the bimodal probability distribution of the set of reactions.

4.1 Results of the flowchart

In this subsection we will see the result obtained by applying our method to infer an MP system modelling the stochastic bistability of the Schlögel model.

Obtaining data.

The Schlögel's reaction has been simulated by considering the three different initial states (5) to generate the time series of species x. Then, we clustered the data of the different behaviours in four sets and we calculated the average number of molecules during all the steps of the time evolution. In particular, these were sampled at regular time intervals to mimic experimental measurements.

Inferring intermediate MP models.

First, we computed reaction fluxes by applying the Log-Gain theory with different plausible covering sets [4], concluding that r_3 has a constant flux. This result is in accordance with the nature of this reaction, which has a buffered reactant. After that, considering the four behaviours, we applied the least-squares theory to infer four sets of flux regulation functions, that is, to obtain four MP grammars. In particular, according to a correlation analysis, and given that a and b are buffered species, we assumed that each functions $\varphi_j(q)$, $j = 1, 2, \ldots, 4$, can be seen as:

$$\varphi_j(q) = \begin{cases} \alpha_j + \beta_j x & \text{if } j = 1, 2, 4 \\ \alpha_j & \text{if } j = 3. \end{cases} \tag{6}$$

In this way, we obtained four MP systems, reported in Tables 2, 3, 4 and 5, which characterize the different behaviours showed in Figures 2 and 3.

Reaction	Maps
$r_1 : a + 2x \rightarrow 3x + a$	$\varphi_1 = 3.4710 \cdot 10^{-3} + 4.1533 \cdot 10^{-1}x$
$r_2 : 3x \rightarrow 2x$	$\varphi_2 = 4.7307 \cdot 10^{-1} + 4.0487 \cdot 10^{-1}x$
$r_3 : b \rightarrow x + b$	$\varphi_3 = 0.87437$
$r_4 : x \rightarrow \lambda$	$\varphi_4 = 1.2542 \cdot 10^{-4} + 1.5008 \cdot 10^{-2}x$

Table 2. Flux regulation functions approximating the behaviour of Schlögel's reaction computing by the SSA, starting from the initial state $\#X[0] = (1 \cdot 10^5, 2 \cdot 10^5, 0)$, that is, the behaviour of the species x showed in the left part of Figure 2.

Reaction	Maps
$r_1 : a + 2x \rightarrow 3x + a$	$\varphi_1 = -2.1926 \cdot 10^{-13} + 2.5331 \cdot 10^{-4}x$
$r_2 : 3x \rightarrow 2x$	$\varphi_2 = 7.1050 \cdot 10^{-1} - 2.9884 \cdot 10^{-4}x$
$r_3 : b \rightarrow x + b$	$\varphi_3 = 0.87437$
$r_4 : x \rightarrow \lambda$	$\varphi_4 = 2.5415 \cdot 10^{-13} + 8.4015 \cdot 10^{-4}x$

Table 3. Flux regulation functions approximating the behaviour of Schlögel's reaction computing by the SSA, considering the initial state $\#X[0] = (1 \cdot 10^5, 2 \cdot 10^5, 800)$, that is, the behaviour of the species x showed in the right part of Figure 2.

Reaction	Maps
$r_1 : a + 2x \rightarrow 3x + a$	$\varphi_1 = -1.2646 \cdot 10^{-15} + 8.5151 \cdot 10^{-4}x$
$r_2 : 3x \rightarrow 2x$	$\varphi_2 = 8.5511 \cdot 10^{-1} - 1.7333 \cdot 10^{-3}x$
$r_3 : b \rightarrow x + b$	$\varphi_3 = 0.87437$
$r_4 : x \rightarrow \lambda$	$\varphi_4 = 5.7823 \cdot 10^{-15} + 2.8242 \cdot 10^{-3}x$

Table 4. Flux regulation functions approximating the behaviour of Schlögel's reaction dynamics depicted in the left part of Figure 3.

Reaction	Maps
$r_1 : a + 2x \rightarrow 3x + a$	$\varphi_1 = -1.6164 \cdot 10^{-13} + 1.7453 \cdot 10^{-3}x$
$r_2 : 3x \rightarrow 2x$	$\varphi_2 = 8.1220 \cdot 10^{-1} + 1.7921 \cdot 10^{-3}x$
$r_3 : b \rightarrow x + b$	$\varphi_3 = 0.87437$
$r_4 : x \rightarrow \lambda$	$\varphi_4 = -7.8644 \cdot 10^{-16} + 6.3066 \cdot 10^{-5}x$

Table 5. Flux regulations functions approximating the behaviour of Schlögel's reaction dynamics depicted in the right part of Figure 3.

Inferring a bistable MP system.

In this phase we used the four intermediate MP models as starting points to obtain an MP system $\mathcal{M}$ describing the bistable behaviours of the Schlögel's model. In particular, we followed these steps: *i)* we computed the dynamics of the four MP systems to obtain four time-series of x and u_1, u_2, u_3 and u_4, respectively, *ii)* we approximated the flux regulation functions, starting from the time-series obtained in the previous step, by using the least square theory, *iii)* we analyzed the inferred functions, and *iii)* finally we obtained the final MP grammar of $\mathcal{M}$.

The Schlögel's model has four reactions, each of which is equipped with a function describing how that reaction contributes to the change of the number of molecules of x: $\varphi_1(q)$, $\varphi_2(q)$, $\varphi_3(q)$, and $\varphi_4(q)$, respectively.

Since the Schlögel's model is a one-variable system, an MP system modelling it can be expressed, considering the stoichiometry of each reaction, as the sum of the four fluxes:

$$x[i+1] - x[i] = \sum_{i=1}^{4} (r_i^+ - r_i^-)\varphi_i(q) = f(q). \tag{7}$$

To describe a bistable behaviour, equation (7) needs an unstable steady state to separate the attractor regions of two stable steady states [24], so we need a function $f(q)$, called the *global flux function* of x, having at least three steady states to realize an MP system describing the bistability of the studied process.

The simplest function $f(q)$ with three zeros is the cubic polynomial. Therefore, since the fluxes of reaction r_3 are constant, we assumed the following forms for the flux regulation functions of $\mathcal{M}$:

$$\varphi_j(q) = \begin{cases} \alpha_j + \beta_j x + \gamma_j x^2 + \eta_j x^3 & \text{if } j = 1, 2, 4 \\ \alpha_j & \text{if } j = 3 \end{cases} \tag{8}$$

and we applied the least square theory to learn the coefficients of each function, obtaining the MP grammar reported in Table 6, which models the bistable behaviour of the Schlögel's model. To prove this, if we consider such flux regulation functions, $f(q)$ can be reduced as follow:

$$f(q) = c_1 x^2 + c_2 x^3 + c_3 + c_4 x \tag{9}$$

where:

$$\begin{aligned} c_1 &= \sum_{j=1}^{4} \gamma_j = 1.0480 \cdot 10^{-6} & c_2 &= \sum_{j=1}^{4} \eta_j = -1.1130 \cdot 10^{-9} \\ c_3 &= \sum_{j=1}^{4} \alpha_j = 1.9370 \cdot 10^{-2} & c_4 &= \sum_{j=1}^{4} \beta_j = -2.6800 \cdot 10^{-4}. \end{aligned} \tag{10}$$

The set of parameters $\{c_j \mid j = 1, 2, \ldots, 4\}$ is associated with a bistable dynamics, that is, two stable steady states separated by an unstable state. By using the discriminant analysis, we can analyze the nature of the roots of a polynomial. The discriminant of (9) is given by:

$$\Delta = c_1^2 c_4^2 - 4c_2 c_4^3 - 4c_1^3 c_3 - 27c_2^2 c_3^2 + 18 \prod_{j=1}^{4} c_j. \tag{11}$$

For a cubic polynomial we have the following cases: *i)* if $\Delta > 0$ then the polynomial has 3 distinct real roots, *ii)* if $\Delta < 0$ then the polynomial has 1 real root and 2 complex conjugate roots, and *iii)* if $\Delta = 0$ then at least 2 polynomial's roots coincide, and they are all real. The cubic polynomial in (9) has $\Delta = 4.5012 \cdot 10^{-22}$, so such function has three different roots. Moreover, to realize a stable 'on' state the sign of the cubic term needs to be a minus. For three different non-negative steady states a positive quadratic and a negative linear term are needed. In addition if a positive constant is adjoint, then the one-variable system:

$$x[i+1] - x[i] = -k_1 x^3 + k_2 x^2 - k_3 x + k_4, \quad k_i > 0, \quad i = 1, 2, 3, 4. \tag{12}$$

has two positive stable steady states [25].

It is simple to see that equation (9), considering the constants (10), is in accordance with (12). Then the MP grammar of Table 6 models the bistable dynamics of the Schlögel's reaction, which has two positive stable steady states.

However, we saw that, starting from some initial states, we obtained negative fluxes. Therefore, we applied the last phase of our flowchart to obtain a final "good" set of functions.

Reaction	Maps
$r_1 : a + 2x \to 3x + a$	$\varphi_1 = 47.133 - 3.5489 \cdot 10^{-1}x + 7.8941 \cdot 10^{-4}x^2 - 5.4356 \cdot 10^{-7}x^3$
$r_2 : 3x \to 2x$	$\varphi_2 = 46.103 - 3.4186 \cdot 10^{-1}x + 7.6346 \cdot 10^{-4}x^2 - 5.2949 \cdot 10^{-7}x^3$
$r_3 : b \to x + b$	$\varphi_3 = 0.87437$
$r_4 : x \to \lambda$	$\varphi_4 = 1.885 - 1.2762 \cdot 10^{-2}x + 2.4902 \cdot 10^{-5}x^2 - 1.2957 \cdot 10^{-8}x^3$

Table 6. Bistable MP grammar modelling the Schlögel's reaction.

Improving flux regulation functions

First, by considering equation (12), we assumed that, since $c_i > 0, i = 1, 3$ then the monomials $c_1 x^2$ and the constant c_3, are associated with reactions producing x. Since r_3 has b as reactant, which is a buffered species, and $u_3[t]$ is constant for $t = 0, 1, \ldots$, then $\varphi_3(q) = c_3$. This implies that $\varphi_1(q) = c_1 x^2$, which is in accordance to the fact that r_1 is a double-molecular reaction. Similarly, $c_i < 0, i = 2, 4$, therefore $c_2 x^3$ and $c_4 x$ can be seen as functions regulating the fluxes of reactions consuming x. Since, r_2 and r_4 are tri-molecular and an one-molecular reaction respectively, we assumed that $\varphi_2(q) = c_2 x^3$ and $\varphi_4(q) = c_4 x$. In this way, we obtained the MP grammar reported in Table 7, which computes the same dynamics of the grammar of Table 7, but having flux time-series positive for each initial states.

5 Conclusion and ongoing work

Schlögel's model is an example of chemical reaction system which exhibits bistability. Bistable behaviour can be found in many biological networks, including heart models, visual perception and gene networks.

Reaction	Maps
$r_1 : a + 2x \rightarrow 3x + a$	$\varphi_1 = 1.0480 \cdot 10^{-6} x^2$
$r_2 : 3x \rightarrow 2x$	$\varphi_2 = 1.1130 \cdot 10^{-9} x^3$
$r_3 : b \rightarrow x + b$	$\varphi_3 = 1.9370 \cdot 10^{-2}$
$r_4 : x \rightarrow \lambda$	$\varphi_4 = 2.6800 \cdot 10^{-4} x$

Table 7. Bistable MP grammar modelling the Schlögel's reaction.

Owing to the ubiquity and importance of switching behaviours, it is important to have comprehensive mathematical models of bistable chemical reaction systems. In particular, there is a lag in the development of models for bistable systems starting from experimental data. This is due to the lack of detailed knowledge of biochemical reactions and kinetic rates.

In this work, we used the Schlögel's model as an example to study the applicability of the MP Systems to infer mathematical models describing observed bistable (multistable) dynamics. The theoretical background of this approach comes from the Log-Gain theory for MP systems, which links observed time-series to the MP systems for simulating and analyzing dynamics of phenomena in living cells. Compared with approaches based on stochastic models and mass action law, our approach allows to obtain some insights into the logic governing a bistable phenomenon starting from observations of such a phenomenon.

Starting from stochastic dynamics of the Schlögel's model, we saw the possibility to obtain an MP system describing the bistability of such dynamics. Since, different studies indicates that noise plays an important roles in the switching of bistable systems, the results of this work suggests that the proposed approach is a very promising one for inferring and studying bistable and multistable dynamics of biological systems, also when kinds of noise are present. Moreover, this approach could be very useful in the cases of complex reaction networks, for which data availability and regulatory information can not provide a comprehensive picture of the role of the diverse reactions in the toggle switch transition.

Ongoing research is focused on the application of the proposed approach to infer bistable systems inspired from biology and chemistry and analyze the logic governing these systems. In particular, since bistable switches are common motifs in genetic regulatory networks, we have a mind to apply our procedure for modelling a naturally occurring switch from relatively few experimental data points, yielding a model suited: *i)* to dynamical simulation, *ii)* to give predictions of unmeasured proteins and genes of the analyzed network, *iii)* to analyze the effects of noise and perturbations which can afflict the network, *iv)* and to develop robust mathematical models which could represent prototypes of synthetic biological systems.

References

1. L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 81–126. Springer, Berlin, 2006.
2. T. Eissing, H. Conzelmann, E. D. Gilles, F. Allgower, E. Bullinger, and P. Scheurich. Bistability analyses of a caspase activation model for receptor-induced apoptosis. *J. Biol. Chem.*, 279(35):36892–36897, 2004.
3. J. E. Ferrell. Self-perpetuating states in signal transduction: positive feedback, double-negative feedback and bistability. *Current Opinion in Cell Biology*, 14(2):140 – 148, 2002.
4. G. Franco, V. Manca, and R. Pagliarini. Regulation and covering problems in MP systems. In *LNCS*, volume 5957, pages 242–251. Springer-Verlag, 2010.
5. D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.
6. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
7. L. Giot, J. S. Bader, C. Brouwer, A. Chaudhuri, B. Kuang, Y. Li, Y. L. Hao, C. E. Ooi, B. Godwin, E. Vitols, G. Vijayadamodar, P. Pochart, H. Machineni, M. Welsh, Y. Kong, B. Zerhusen, R. Malcolm, Z. Varrone, A. Collis, M. Minto, S. Burgess, L. McDaniel, E. Stimpson, F. Spriggs, J. Williams, K. Neurath, N. Ioime, M. Agee, E. Voss, K. Furtak, R. Renzulli, N. Aanensen, S. Carrolla, E. Bickelhaupt, Y. Lazovatsky, A. DaSilva, J. Zhong, C. A. Stanyon, R. L. Finley, K. P. White, M. Braverman, T. Jarvie, S. Gold, M. Leach, J. Knight, R. A. Shimkets, M. P. McKenna, J. Chant, and J. M. Rothberg. A protein interaction map of drosophila melanogaster. *Science*, 302(5651):1727–1736, 2003.
8. J. Hasty, D. McMillen, F. Isaacs, and J. J. Collins. Computational studies of gene regulatory networks: in numero molecular biology. *Nature reviews. Genetics*, 2(4):268–279, 2001.
9. T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proceedings of the National Academy of Sciences of the United States of America*, 98(8):4569–4574, 2001.
10. M. Kaern, T. C. Elston, W. J. Blake, and James J. C. Stochasticity in gene expression: from theories to phenotypes. *Nature Reviews Genetics*, 6(6):451–464, 2005.
11. N. Kellershohn and M. Laurent. Prion diseases: dynamics of the infection and properties of the bistable transition. *Biophys Journal*, 81(5):2517–2529, 2001.
12. D. Kim, O. Rath, W. Kolch, K. H. Cho, , and . A hidden oncogenic positive feedback loop caused by crosstalk between wnt and erk pathways. *Oncogene*, 26:4571–4579, 2007.
13. S. Macnamara, K. Burrage, and R.B. Sidje. Multiscale modeling of chemical kinetics via the master equation. *Multiscale Modeling & Simulation*, 6(4):1146–1168, 2008.
14. V. Manca. The Metabolic Algorithm: Principles and applications. *Theoretical Computer Science*, 404:142–157, 2008.
15. V. Manca. Fundamentals of metabolic P systems. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 16. Oxford University Press, 2009.

16. V. Manca. Log-gain principles for metabolic P systems. In A. Condon, D. Harel, J.N. Kok, A. Salomaa, and E. Winfree, editors, *Algorithmic Bioprocesses*, Natural Computing Series, chapter 28. Springer, 2009.
17. V. Manca. From P to MP Systems. In *LNCS*, volume 5957, pages 74–94. Springer-Verlag, 2010.
18. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biochemical phenomena. In *LNCS*, volume 3365, pages 63–84. Springer, 2005.
19. V. Manca, R. Pagliarini, and S. Zorzan. A photosynthetic process modelled by a metabolic p system. *Natural Computing*, 8(4):847–864, 2009.
20. G. Păun. *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
21. F. Schlögl. Chemical reaction models for non-equilibrium phase transitions. *Zeitschrift für Physik A Hadrons and Nuclei*, 253(2):147–161, 1972.
22. B. M. Slepchenko and M. Terasaki. Bio-switches: what makes them robust? *Current Opinion in Genetics & Development*, 14(4):428 – 434, 2004.
23. J. W. Veening, W. K. K. Smits, and O. P. Kuipers. Bistability, epigenetics, and bet-hedging in bacteria. *Annual review of microbiology*, 62(1):193–210, 2008.
24. M. Vellela and H. Qian. Stochastic dynamics and non-equilibrium thermodynamics of a bistable chemical system: the schlögl model revisited. *Journal of The Royal Society Interface*, 39(6): 925–940, 2009.
25. T. Wilhelm. The smallest chemical reaction system with bistability. *BMC Systems Biology*, 3(1):90–98, 2009.

Solving Problems in a Distributed Way in Membrane Computing: dP Systems*

Gheorghe Păun[1,2], Mario J. Pérez-Jiménez[2]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: `gpaun@us.es, marper@us.es`

Summary. Although P systems are distributed parallel computing devices, no explicit way of handling the input in a distributed way in this framework was considered so far. This note proposes a distributed architecture (based on cell-like P systems, with their skin membranes communicating through channels as in tissue-like P systems, according to specified rules of the antiport type), where parts of a problem can be introduced as inputs in various components and then processed in parallel. The respective devices are called dP systems, with the case of accepting strings called dP automata. The communication complexity can be evaluated in various ways: *statically* (counting the communication rules in a dP system which solves a given problem), or *dynamically* (counting the number of communication steps, of communication rules used in a computation, or the number of objects communicated). For each measure, two notions of "parallelizability" can be introduced. Besides (informal) definitions, some illustrations of these idea are provided for dP automata: each regular language is "weakly parallelizable" (i.e., it can be recognized in this framework, using a constant number of communication steps), and there are languages of various types with respect to Chomsky hierarchy which are "efficiently parallelizable" (they are parallelizable and, moreover, are accepted in a faster way by a dP automaton than by a single P automaton). Several suggestions for further research are made.

1 Introduction

P systems are by definition distributed parallel computing devices, [11], [12], [17], and they can solve computationally hard problems in a feasible time, [13], but this efficiency is achieved by a trade-off between space and time, based on the

* The present paper is in press in *Int. J. of Computers, Communication and Control*, Vol. V, No. 2, 2010.

possibility of generating an exponential workspace in a linear time, by means of biologically inspired operations, such as membrane division and membrane creation. However, no class of P systems was proposed where a hard problem can be solved in a distributed parallel way after splitting the problem in parts and introducing these subproblems in components of a P system which can work on these subproblems in parallel and produce the solution to the initial problem by interacting/communicating among each other (like in standard distributed computer science). In particular, no communication complexity, in the sense of [2], [9], [16], was considered for P systems, in spite of the fact that computation (time) complexity is very well developed, [13], and also space complexity was recently investigated, [14]. Some proposals towards a communication complexity of P systems were made in [1], but mainly related to the communication effort in terms of symport/antiport rules used in so-called evolution-communication P systems of [5]. (Note that in communication complexity theory the focus is not on the time efficiency of solving a problem, but the parties involved in the computation just receive portions of the input, in general, distributed in a balanced manner, "as fair as possible" – this distribution introduces an inherent difficulty in handling the input – and then mainly the complexity of the communication needed to parties to handle this input is investigated.)

This note tries to fill in this gap, proposing a rather natural framework for solving problems in a distributed way, using a class of P systems which mixes ingredients already existing in various much investigated types of P systems. Namely, we consider P systems with inputs, in two variants: (i) like in P automata, [6], [10], where a string of symbols is recognized if those symbols are brought into the system from the environment and the computation eventually halts (it is important to note that the string is "read" during the computation, not *before* it), and (ii) in the usual manner of complexity investigations, [13], where an instance of a decision problem is introduced in a P system in the form of a multiset of symbols (this operation takes no time, the computation starts *after* having the code of the problem inside), and the system decides that instance in the end of a computation which sends to the environment one of the special objects `yes` or `no`. Several such systems, no matter of what type, are put together in a complex system which we call *dP system* (from "distributed P system"); the component systems communicate through channels linking their skin membranes, by antiport rules as in tissue-like P systems. When accepting strings by dP systems with P automata as components, the device is called a *dP automaton.*

Such an architecture was already used, with specific ingredients, for instance, in the investigations related to eco-systems, where "local environments" are necessary to be delimited and communication possibilities exist, linking them; details can be found in the recent paper [4].

The way to use a dP system is obvious: a problem Q is split into parts $q_1, q_2, \ldots, q_n$, which are introduced in the n components of the dP system (as in P automata or as in decision P systems), these n systems work separately on their problems, and communicate to each other according to the skin-to-skin rules.

The solution to the problem Q is provided by the whole system (by halting – in the case of accepting strings, by sending out one of the objects `yes` or `no`, etc.). Like in communication complexity, [9], we request the problem to be distributed in a *balanced* way among the components of the dP system, i.e., in "as equal as possible" parts (also an *almost balanced* way to distribute the input among two processors is considered in [9] – no partner takes more than two thirds of the input – which does not seem very natural to be extended to the general case, of n processors).

Several possibilities exist for defining the communication complexity of a computation. We follow here the ideas of [1], and introduce three measures: the number of steps of the computation when a communication rule is used (such a step is called communication step), the number of communication rules used during a computation, and the number of objects transferred among components (by communication rules) during a computation. All these three measures are dynamically defined; we can also consider a static parameter, like in descriptional complexity of Chomsky languages (see a survey in [8]), i.e., the number of communication rules in a dP system.

A problem is said to be "weakly parallelizable" with respect to a given (dynamical) communication complexity measure if it can be split in a balanced way, introduced in the dP system, and solved using a number of communication steps bounded by a constant given in advance; a problem is "efficiently parallelizable" if it is weakly parallelizable and can be solved by a dP system in a more efficient way than by a single P system; more precise definitions are given in the next sections of the paper.

Various possibilities exist, depending on the type of systems (communicating systems, e.g., based on symport/antiport rules, systems with active membranes, catalytic systems, etc.) and the type of problem we consider (accepting strings, decision problems, numerical problems, etc.).

In this note we only sketch the general formal framework and give an illustration, for the case of accepting strings as in P automata. We only show here that all regular languages are weakly parallelizable (only one communication step suffices, hence the weak parallizability holds with respect to all three dynamical measures), and that there are regular, context-free non-regular, context-sensitive non-context-free languages which are efficiently parallelizable with respect to the first two dynamical measures mentioned above (in view of the results in [9], there are linear languages which are not efficiently parallelizable with respect to the number of communicated objects/bits among components).

If we use extended systems (a terminal alphabet of objects is available) and the communication channels among the components of a dP automaton are controlled, e.g., by states, as in [7], or created during the computation, as in [3], then the power of our devices increases considerably: all recursively enumerable languages are weakly parallelizable in this framework.

Many research problems remain to be explored, starting with precise definitions for given classes of P systems, continuing with the study of usefulness of this strategy for solving computationally hard problems (which problems are weakly/efficiently parallelizable and which is the obtained speed-up for them?), and ending with a communication complexity theory of dP systems, taking into account all measures of complexity mentioned above (for the number of objects communicated among components, which corresponds to the number of bits considered in [9], we can transfer here the general results from communication complexity – note however that in many papers in this area one deals with 2-party protocols, while in our framework we want to have an n-party set-up, and that we are also interested in the time efficiency of the distributed and parallel way of solving a problem).

2 dP Systems – A Preliminary Formalization

The reader is assumed familiar with basics of membrane computing, e.g., from [11], [12], and of formal language theory, e.g., from [15], hence we pass directly to introducing our proposal of a distributed P system. The general idea is captured in the following notion.

A *dP scheme* (of degree $n \geq 1$) is a construct

$$\Delta = (O, \Pi_1, \ldots, \Pi_n, R),$$

where:

1. O is an alphabet of objects;
2. $\Pi_1, \ldots, \Pi_n$ are cell-like P systems with O as the alphabet of objects and the skin membranes labeled with $s_1, \ldots, s_n$, respectively;
3. R is a finite set of rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n$, $i \neq j$, and $u, v \in O^*$, with $uv \neq \lambda$; $|uv|$ is called the *weight* of the rule $(s_i, u/v, s_j)$.

The systems $\Pi_1, \ldots, \Pi_n$ are called *components* of the scheme Δ and the rules in R are called *inter-components communication rules.* Each component can take an input, work on it, communicate with other components (by means of rules in R), and provide the answer to the problem in the end of a halting computation. (A delicate issue can appear in the case of components which can send objects to the environment and bring objects from the environment – this happens, for instance, for symport/antiport P systems; in this case we have to decide whether or not the components can exchange objects by means of the environment, or the only permitted communication is done by means of the rules in R. For instance, a "local environment" for each component can be considered, disjoint from the "local environments" of other components, thus preventing the interaction of components by means of other rules than those in R. Actually, the rules in R themselves can be defined between these "local environments" – which is a variant worth to explore. We point out here that also the need of a "local environment" has appeared in the applications of membrane computing to eco-systems investigations, see [4] and its references.)

Now, we can particularize this notion in various ways, depending on the type of systems $\Pi_i, 1 \leq i \leq n$, and the type of problems we want to solve.

For instance, we can define *dP systems with active membranes*, as dP schemes as above, with the components being P systems with active membranes, each of them having a membrane designated as the input membrane. Having a decision problem – consider, e.g., SAT for n variables and m clauses – we can split a given instance of it in parts which are encoded in multisets which are introduced in the components of the dP system. For example, we can introduce the code of each separate clause in a separate component of the dP system. The components start to work, each one deciding its clause, and in the end they communicate to each other the result; if one of the components will find that all m clauses are satisfied, then the whole SAT formula is satisfied. Intuitively, this is a faster way than deciding the formula by means of a single P system with active membranes – but a crucial aspect has been neglected above: in order to say that the formula is satisfied, all the m clauses should be satisfied *by the same truth-assignment*, and this supposes that the m components communicate to each other also which is the assignment which turns true the clauses. That is, besides the usual time complexity of solving the problem we have now to consider the cost of communication among the components and the trade-off between these two parameters should be estimated.

Another interesting case, which will be briefly investigated in the subsequent section, is that of accepting strings in the sense of P automata, [6], [10]; we will come back immediately to this case.

On the other hand, we have several possibilities for estimating "the cost of communication", and we adapt here the ideas from [1].

Let us consider a dP system Δ, and let $\delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ be a halting computation in Δ, with w_0 being the initial configuration. Then, for each $i = 0, 1, \ldots, h-1$ we can write:

$ComN(w_i \Longrightarrow w_{i+1}) = 1$ if a communication rule is used in this transition, and 0 otherwise,
$ComR(w_i \Longrightarrow w_{i+1}) =$ the number of communication rules used in this transition,
$ComW(w_i \Longrightarrow w_{i+1}) =$ the total weight of the communication rules used in this transition.

These parameters can then be extended in the natural way to computations, results of computations, systems, problems/languages. We consider below the case of accepting strings (by $L(\Delta)$ we denote the language of strings accepted by Δ): for $ComX \in \{ComN, ComR, ComW\}$ we define

$ComX(\delta) = \sum_{i=0}^{h-1} ComX(w_i \Longrightarrow w_{i+1})$, for $\delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ a halting computation,
$ComX(w, \Delta) = \min\{ComX(\delta) \mid \delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ is a computation in Δ which accepts the string $w\}$,
$ComX(\Delta) = \max\{ComX(w, \Delta) \mid w \in L(\Delta)\}$,
$ComX(L) = \min\{ComX(\Delta) \mid L = L(\Delta)\}$.

Similar definitions can be considered for more general decidability problem than accepting strings, then complexity classes can be defined. We do not enter here into details for this general case; in the next section we will briefly consider the specific case of dP automata and of languages.

The previously sketched approach should be investigated in more details. Which is the speed-up for a given problem or class of problems? Clearly, $ComN(\alpha) \leq ComR(\alpha) \leq ComW(\alpha)$, for all valid α. Moreover, in one communication step one can use arbitrarily many communication rules, which therefore move from a component to another one arbitrarily many objects. Anyway, independently of the communication cost, presumably, only a linear speed-up can be obtained by splitting the problem in a given number of parts. Are there problems which however cannot be solved in this framework in a faster way than by using a single P system (with active membranes) provided that the communication cost is bounded (e.g., using communication rules in R only for a constant number of times)? Which is the communication complexity for a given problem or class of problems? Finding suggestive examples can be a first step in approaching such issues.

A case study will be considered in the next section, not for dP systems with active membranes (which, we believe, deserve a separate and detailed examination), but for a distributed version of P automata.

3 dP Automata

We consider now the distributed version of P automata, [6], [10], which are symport/antiport P systems which accept strings: the sequence of objects (because we work with strings and symbol objects, we use interchangeably the terms "object" and "symbol") imported by the system from the environment during a halting computation is the string accepted by that computation (if several objects are brought in the system at the same time, then any permutation of them is considered as a substring of the accepted string; a variant, considered in [6], is to associate a symbol to each multiset and to build a string by such "marks" attached to the imported multisets). The accepted string can be introduced in the system symbol by symbol, in the first steps of the computation (if the string is of length k, then it is introduced in the system in the first k steps of the computation – the P automaton is then called *initial*), or in arbitrary steps. Of course, the initial mode is more restrictive – but we do not enter here into details.

As a kind of mixture of the ideas in [6] and [10] for defining the accepted language, we can consider extended P automata, that is, with a distinguished alphabet of objects, T, whose elements are taken into account when building the accepted string (the other objects taken by the system from the environment are ignored). Here, however, we work with non-extended P automata.

A *dP automaton* is a construct

$$\Delta = (O, E, \Pi_1, \ldots, \Pi_n, R),$$

where $(O, \Pi_1, \ldots, \Pi_n, R)$ is a dP scheme, $E \subseteq O$ (the objects available in arbitrarily many copies in the environment), $\Pi_i = (O, \mu_i, w_{i,1}, \ldots, w_{i,k_i}, E, R_{i,1}, \ldots, R_{i,k_i})$ is a symport/antiport P system of degree k_i (without an output membrane), with the skin membrane labeled with $(i, 1) = s_i$, for all $i = 1, 2, \ldots, n$.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over O if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, using the symport/antiport rules as well as the inter-components communication rules, in the non-deterministically maximally parallel way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

The dP automaton is synchronized, a universal clock exists for all components, marking the time in the same way for the whole dP automaton. It is also important to note that we work here in the non-extended case, all input symbols are recorded in the string. In this way, at most context-sensitive languages can be recognized.

The three complexity measures $ComN, ComR, ComW$ defined in the previous section can be directly introduced for dP automata (and they were formulated above for this case). With respect to them, we can consider two levels of parallelizability.

A language $L \subseteq V^*$ is said to be (n, m)-*weakly ComX parallelizable*, for some $n \geq 2, m \geq 1$, and $X \in \{N, R, W\}$, if there is a dP automaton Δ with n components and there is a finite subset F_Δ of L such that each string $x \in L - F_\Delta$ can be written as $x = x_1 x_2 \ldots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the string x is accepted by Δ by a halting computation δ such that $ComX(\delta) \leq m$. A language L is said to be *weakly ComX parallelizable* if it is (n, m)-weakly $ComX$ parallelizable for some $n \geq 2, m \geq 1$.

Two conditions are here important: (i) the string is distributed in equal parts, modulo one symbol, to the components of the dP automaton, and (ii) the communication complexity, in the sense of measure $ComX$, is bounded by the constant m.

We have said nothing before about the length of the computation. That is why we also introduce a stronger version of parallelizability.

A language $L \subseteq V^*$ is said to be (n, m, k)-*efficiently ComX parallelizable*, for some $n \geq 2, m \geq 1, k \geq 2$, and $X \in \{N, R, W\}$, if it is (n, m) weakly ComX parallelizable, and there is a dP automaton Δ such that

$$\lim_{x \in L, |x| \to \infty} \frac{time_\Pi(x)}{time_\Delta(x)} \geq k,$$

for all P automata Π such that $L = L(\Pi)$ ($time_\Gamma(x)$ denotes here the smallest number of steps needed for the device Γ to accept the string x). A language L is said to be *efficiently ComX parallelizable* if it is (n, m, k)-efficiently $ComX$ parallelizable for some $n \geq 2, m \geq 1$, $k \geq 2$.

Note that in the case of dP automata, the duration of a computation may also depend on the way the string is split in substrings and introduced in the components of the system; in a natural way, one of the most efficient distribution of the string and shortest computation are chosen. Of course, as larger the constant k as better.

Moreover, while $time_{\Delta}(x)$ is just given by means of a construction of a suitable dP automaton Δ, $time_{\Pi}(x)$ should be estimated *with respect to all P automata* Π.

An example is worth considering in order to illustrate this definition. Let us examine the dP system from Figure 1 – the alphabet of objects is $O = \{a, b, c, d, c_1, c_2, \#\}$, and $E = \{a, b\}$.

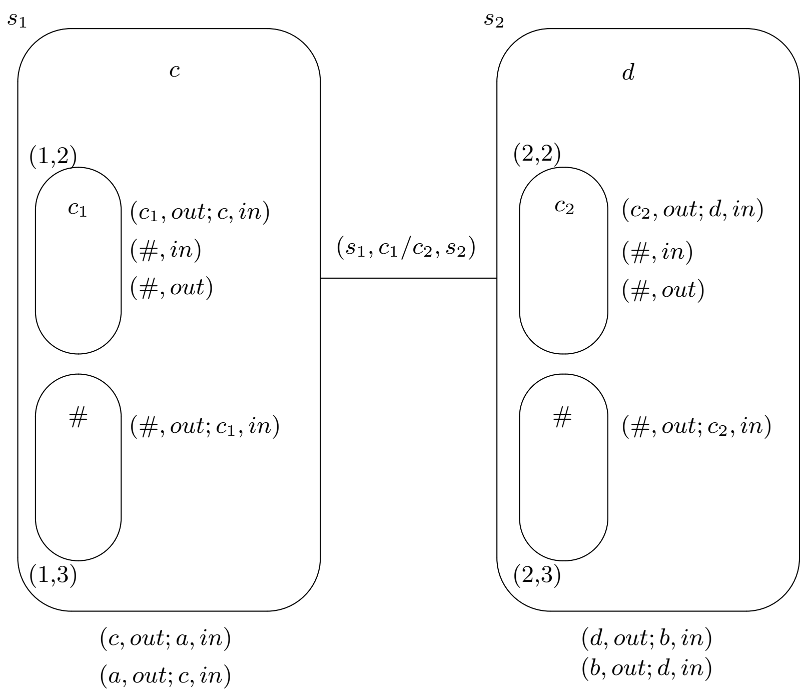


Fig. 1. An example of a dP automaton

Clearly, component Π_1 (in the left hand side of the figure) can only bring objects a, c inside, and component Π_2 (in the right hand side of the figure) can only bring objects b, d inside. In each step, only one of a, c, alternately, enters Π_1 and only one of b, d, alternately, enters Π_2 (note that we do not need objects c, d to be present initially in the environment, while one copy of each a and b is sufficient). The computation of each component can stop only by "hiding" the "carrier objects" c, d inside an inner membrane, and this means releasing c_1 in Π_1 and c_2 in Π_2. If these objects are not released at the same time in the two components, so that the exchange rule $(s_1, c_1/c_2, s_2)$ can be used, then, because of the maximal parallelism, the object c_1 should enter membrane (1,3), and object c_2 should enter membrane (2,3); in each case, the trap-object # is released, and the computation never stops: the object # oscillates forever across membrane (1,2) in Π_1 and across membrane (2,2) in Π_2.

Consequently, the two strings accepted by the two components of Δ should have the same length, that is the language accepted by the system is

$$L(\Delta) = \{(ac)^s(bd)^s \mid s \geq 0\}.$$

Note the crucial role played here by the fact that the system is synchronized, and that a computation which accepts a string $x_s = (ac)^s(bd)^s$, hence of length $4s$, lasts $2s + 2$ steps ($2s$ steps for bringing objects inside, one step when objects c, d are introduced in an inner membrane, and one inter-components communication step), with one of these steps being a communication between components.

Obviously, if we recognize a string $x_s = (ac)^s(bd)^s$ as above by means of a usual symport/antiport P system, then, because no two symbols of the string can be interchanged, no two adjacent symbols can be introduced in the system at the same step, hence the computation lasts at least as many steps as the length of the string, that is, $4s$. This shows that our language is not only $(2, r)$-weakly $ComX$ parallelizable, but also $(2, r, 2)$-efficiently $ComX$ parallelizable, for $(r, X) \in \{(1, N), (1, R), (2, W)\}$.

This conclusion is worth formulating as a theorem.

Theorem 1. *The language* $L = \{(ac)^s(bd)^s \mid s \geq 0\}$ *is efficiently* $ComX$ *parallelizable, for all* $X \in \{N, R, W\}$.

Note that this language is not regular (but it is linear, hence also context-free).

The previous construction can be extended to dP automata with three components: Π_1 inputs the string $(ac)^s$, Π_2 inputs $(bd)^s$, and Π_3 inputs $(ac)^s$, then Π_1 produces the object c_1, Π_2 produces two copies of c_2, and Π_3 produces the object c_3. Now, c_1 is exchanged for one copy of c_2 from Π_2 and c_3 for the other copy, otherwise the computation never stops. The recognized language is $\{(ac)^s(bd)^s(ac)^s \mid s \geq 0\}$.

This language is not context-free, hence we have:

Theorem 2. *There are context-sensitive non-context-free languages which are efficiently* $ComX$ *parallelizable, for all* $X \in \{N, R, W\}$.

The previous two theorems show that the distribution, in the form of dP systems, is useful from the time complexity point of view, although only one communication step is performed and only one communication rule is used at that step. Moreover, the proofs of the two theorems show that, in general, languages consisting of strings with two well related halves (but not containing "too much" information in each half of the string, besides the length), are weakly parallelizable, and, if no two adjacent symbols of the strings can be interchanged, then these languages are efficiently parallelizable.

We have said nothing above about regular languages – this is the subject of the next section.

4 All Regular Languages are Weakly Parallelizable

The assertion in the title of this section corresponds to Theorem 2.3.5.1 in [9], which states that for each regular language there is a constant k which bounds its (2-party) communication complexity. The version of this result in terms of weak $ComX$ parallelizability is shown by the following construction. Consider a non-deterministic finite automaton $A = (Q, T, q_0, F, P)$ (set of states, alphabet, initial state, final states, set of transition rules, written in the form $qa \to q'$, for $q, q' \in Q, a \in T$). Without any loss of generality, we may assume that all states of Q are reachable from the initial state (for each $q \in Q$ there is $x \in T^*$ such that $q_0 x \Longrightarrow^* q$ with respect to transition rules in P). We construct the following dP automaton:

$$\begin{aligned}
\Delta &= (O, E, \Pi_1, \Pi_2, R), \text{ where :}\\
O &= Q \cup T \cup \{d\}\\
&\quad \cup \{(q, q') \mid q, q' \in Q\}\\
&\quad \cup \{\langle q, q_f \rangle \mid q \in Q, q_f \in F\}\\
&\quad \cup \{\langle q \rangle \mid q \in Q\},\\
E &= O - \{d\},\\
\Pi_1 &= (O, [_{s_1}[_{1,2}\ ]_{1,2}\ ]_{s_1}, q_0, \lambda, E, R_{s_1}, R_{1,2}),\\
R_{s_1} &= \{(q, out; q'a, in) \mid qa \to q' \in P\}\\
&\quad \cup \{(q, out; \langle q' \rangle a, in) \mid qa \to q' \in P\},\\
R_{1,2} &= \{(\langle q \rangle, in),\ (\langle q \rangle, out) \mid q \in Q\},
\end{aligned}$$

$$\begin{aligned}
\Pi_2 &= (O, [_{s_2}\ ]_{s_2}, d, E, R_{s_2}),\\
R_{s_2} &= \{(d, out; (q, q')a, in) \mid qa \to q' \in P, q \in Q\}\\
&\quad \cup \{((q, q'), out; (q, q'')a, in) \mid q'a \to q'' \in P, q \in Q\}\\
&\quad \cup \{((q, q'), out; \langle q, q_f \rangle a, in) \mid q'a \to q_f \in P, q \in Q, q_f \in F\},\\
R &= \{(s_1, \langle q \rangle / \langle q, q_f \rangle, s_2) \mid q \in Q, q_f \in F\}.
\end{aligned}$$

The first component analyzes a prefix of a string in $L(A)$, the second component analyzes a suffix of a string in $L(A)$, first guessing a state $q \in Q$ from which the automaton starts its work. At some moment, the first component stops bringing objects inside by taking from the environment a symbol $\langle q' \rangle$ for some $q' \in Q$, reached after parsing the prefix of the string in $L(A)$. This object will pass repeatedly across the inner membrane of Π_1. The second component can stop if a state q' is reached in the automaton A for which no rule $q'a \to q''$ exists in P (and then Δ never stops, because its first component never stops), or after reaching a state in F, hence introducing an object of the form $\langle q, q_f \rangle$ for some $q_f \in F$. Note that q is the state chosen initially and always stored in the first position of objects (q_1, q_2) used by Π_2. The computation can halt only by using a communication rule from R, and this is possible only if $q = q'$ – the first component has reached the state of A which was the state from which the second component started its work. Consequently, the concatenation of the two strings introduced in the system by the two components is a string from $L(A)$. Thus, the language $L(A)$ is weakly parallelizable.

Now, consider a regular language such that no two adjacent symbols in a string can be permuted (take an arbitrary regular language L over an alphabet V and a morphism $h : V^* \longrightarrow (V \cup \{c\})^*$, where c is a symbol not in V, such that $h(a) = ac$ for each $a \in V$). Then, clearly, if the two strings accepted by the two components of the dP automaton Δ are of equal length (note that the strings of $h(L)$ are of an even length), then the time needed to Δ to accept the whole string is (about) half of the time needed to any P automaton Π which accepts the same language. This proves that the language $h(L)$ is efficiently parallelizable, hence we can state:

Theorem 3. *Each regular language is weakly $ComX$ parallelizable, and there are efficiently $ComX$ parallelizable regular languages, for all $X \in \{N, R, W\}$.*

Of course, faster dP automata can be constructed, if we use more than two components. However, it is not clear whether dP automata with $n+1$ components are always faster than dP automata with n components – this might depend on the structure of the considered language (remember that the distribution of the input string to the components of the dP automaton must be balanced). More specifically, we expect that there are (n, m) weakly parallelizable languages which are not, e.g., $(n + 1, m)$ weakly parallelizable; similar results are expected for efficiently parallelizable languages.

A natural question is how much the result in Theorem 3 can be extended. For instance, is a similar result true for the linear languages, or for bigger families of languages? According to Theorem 2.3.5.4 in [9], this is not true for measures $ComR$ and $ComW$, the recognition of context-free languages (actually, the language L_R at page 78 of [9] is linear) have already the highest communication complexity (in 2-party protocols), a linear one with respect to the length of the string. Thus, the number of communication rules used by a dP automaton during a computation cannot be bounded by a constant. The case of measure $ComN$ remains to be settled: is it possible to have computations with a bounded number of communication steps, but with these steps using an unbounded number of rules? We conjecture that even in this case, languages of the form $\{x\ mi(x) \mid x \in \{a, b\}^*\}$, where $mi(x)$ is the mirror image of x (such a language is minimally linear, i.e., can be generated by a linear grammar with only one nonterminal), are not weakly $ComN$ parallelizable.

Many other questions can be raised in this framework. For instance, we can consider families of languages: (n, m)-weakly $ComX$ parallelizable, weakly $ComX$ parallelizable, (n, m, k)-efficiently $ComX$ parallelizable, and efficiently $ComX$ parallelizable. Which are their properties: interrelationships and relationships with families in Chomsky hierarchy, closure and decidability properties, hierarchies on various parameters, characterizations and representations, etc.

Then, there is another possibility of interest, suggested already above: the static complexity measure defined as the cardinality of R, the set of communication rules. There is a substantial theory of descriptional complexity of (mainly context-free) grammars and languages, see [8], which suggests a lot of research questions starting from $ComS(\varDelta) = card(R)$ (with "S" coming from "static") and extended to languages in the natural way ($ComS(L) = \min\{ComS(\varDelta) \mid L = L(\varDelta)\}$): hierarchies, decidability of various problems, the effect of operations with languages on their complexity, etc.

5 The Power of Controlling the Communication

In the previous sections the communication rules were used as any rule of the system, non-deterministically choosing the rules to be applied, with the communication rules competing for objects with the inner rules of the components, and observing the restriction of maximal parallelism. However, we can distinguish the two types of rules, "internal evolution rules" (transition rules, symport/antiport rules, rules with active membranes, etc.) and communication rules. Then, as in [1], we can apply the rules according to a priority relation, with priority for evolution rules, or with priority for communication rules. Moreover, we can place various types of controls on the communication channel itself. For instance, because the communication rules are antiport rules, we can associate with them promoters or inhibitors, as used in many places in membrane computing.

A still more natural regulation mechanism is to associate *states* with the channels, like in [7]. In this case, the communication rules associated with a pair (i, j) of components $\varPi_i, \varPi_j$ are of the form $(q, u/v, q')$, where q, q' are elements of a given finite set Q of states; initially, the channel is assumed in a given state q_0. A rule as above is applied only if the cannel is in state q – and the antiport rule $(i, u/v, j)$ can be applied; after exchanging the multisets u, v among the two components $\varPi_i, \varPi_j$, the state of the channel is changed to q'.

An important decision should be made in what concerns the parallelism. In [7], the channel rules are used in the sequential mode, but we can also consider two types of parallelism: (i) choose a rule and use it as many times as made possible by the objects in the two components, or (ii) apply at the same time all rules of the form $(q, u/v, q')$ for various u and v (but with the same q and q'), in the non-deterministic maximally parallel way. In the result discussed below, any of these two possibilities works – and the result is somewhat surprising:

Theorem 4. *Any recursively enumerable language L is $(2, 2)$-weakly ComN and ComR parallelizable and has $ComS(L) \leq 2$, with respect to extended dP automata with channel states.*

We do not formally prove this assertion, but we only describe the (rather complex, if we cover all details) construction of the suitable dP automaton.

Take a recursively enumerable language $L \subseteq T^+$, for some $T = \{a_1, a_2, \ldots, a_n\}$. For each string $w \in T^+$, let $val_{n+1}(w)$ be the value of w when considered as a number in base $n+1$, using the digits $a_1, a_2, \ldots, a_n$ interpreted as $1, 2, \ldots, n$, without also using the digit zero. We extend the notation to languages, in the natural way: $val_{n+1}(L) = \{val_{n+1}(w) \mid w \in L\}$. Clearly, L is recursively enumerable if and only if $val_{n+1}(L)$ is recursively enumerable, and the passage from strings w to numbers $val_{n+1}(w)$ can be done in terms of P automata (extended symport/antiport P systems are universal, hence they can simulate any Turing machine; this is one of the places where we need to work with extended systems, as we need copies of a and b – see below – to express the values of strings, and such symbols should be taken from the environment without being included in the accepted strings).

Construct now a dP automaton Δ with two components, Π_1 and Π_2, working as follows. The component Π_1 receives as input a string $w_1 \in T^*$ and Π_2 receives as input a string $w_2 \in T^*$, such that w_1w_2 should be checked whether or not it belongs to the language L. Without loss of generality, we may assume that $|w_1| \in \{|w_2|, |w_2|+1\}$ (we can choose a balanced distribution of the two halves of the string). In the beginning, the state of the channel between the two components is q_0.

Both components start to receive the input symbols, one in each time unit; the component Π_1 transforms the strings w_1 in $val_{n+1}(w_1)$ copies of a symbol a, and Π_2 transforms the string w_2 in $val_{n+1}(w_2)$ copies of a symbol b. When this computation is completed in Π_1, a special symbol, t, is introduced. For this symbol, we provide the communication rule $(q_0, t/\lambda, q_1)$, whose role is to change the state of the channel. We also consider the rule $(q_1, a/\lambda, q_2)$. Using it in the maximally parallel way, all symbols a from Π_1 are moved to Π_2, in one communication step.

Because we have considered w_1 at least of the length of w_2 and we also need two steps for “opening” the channel and for moving the symbols a across it, we are sure that in this moment in Π_2 we have, besides the $val_{n+1}(w_1)$ copies of a, $val_{n+1}(w_2)$ copies of b. The second component takes now these copies of a and b and computes $val_{n+1}(w_1w_2)$, for instance, as the number of copies of an object c. After that, Π_2 checks whether or not $val_{n+1}(w_1w_2) \in val_{n+1}(L)$. If the computation halts, then the string w_1w_2 is accepted, it belongs to the language L.

Note that the dP automaton Δ contains two communication rules (hence $ComS(L) \leq 2$) and that each computation contains two communication steps (hence $ComN(L) \leq 2$), in each step only one rule being used (hence $ComR(L) \leq 2$). These observations complete the proof of the theorem.

Of course, $ComW(\Delta) = \infty$. (Similarly, if we define $ComR$ taking into account the multiplicity of using the rules, then also $ComR$ can be considered infinite – hence the assertion in the theorem remains to be stated only for the measure $ComN$.)

Instead of changing channel states as above, we can assume that the channel itself switches from “virtual” to “actual”, like in population P systems, [3]: the channel is created by object t produced by Π_1, and then used for moving a from

Π_1 to Π_2 by a usual communication rule (which, by definition, is used in the maximally parallel way).

Anyway, the conclusion of this discussion is that the results we obtain crucially depend on the ingredients we use when building our dP systems (as well as on the chosen definitions for complexity measures and types of parallelizability).

6 Closing Remarks

The paper proposes a rather natural way (using existing ingredients in membrane computing, bringing no new, on purpose invented, stuff into the stage) for solving problems in a "standard" distributed manner (i.e., splitting problems in parts, introducing them in various component "computers", and constructing the solution through the cooperation of these components) in the framework of membrane computing. So called dP schemes/systems were defined, and two notions of parallelizability were proposed and briefly investigated for the case of dP automata (accepting strings).

A lot of problems and research topics were suggested. The reader can imagine also further problems, for instance, transferring in this area notions and questions from the communication complexity theory, [9], considering other types of P systems (what about spiking neural P systems, where we have only one type of objects and no antiport-like rules for communicating among components?), maybe using unsynchronized P systems, non-linear balanced input, and so on and so forth. We are convinced that dP systems are worth investigating.

Note. During the recent Brainstorming Week on Membrane Computing, 1-5 of February 2010, Sevilla, Spain, several comments about the definitions and the results of this paper were made, especially by Erzsébet Csuhaj-Varú, György Vaszil, Rudolf Freund, and Marian Kögler. Several continuations of this paper are now in preparation; the interested reader is requested to check the bibliography from [17], in particular, the Brainstorming proceedings volume.

Acknowledgements

This work is supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200. Useful discussions with Henry Adorna are gratefully acknowledged.

References

1. H. Adorna, Gh. Păun, M.J. Pérez-Jiménez: On communication complexity in evolution-communication P systems. Manuscript, 2009.
2. L. Babai, P. Frankl, J. Simon: Complexity classes in communication complexity. *Proc. 27th Annual Symp. Founf. Computer Sci.*, 1986, 337–347.

3. F. Bernardini, M. Gheorghe: Population P systems. *J. Universal Computer Sci.*, 10, 5 (2004), 509–539.
4. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy: A P system based model of an ecosystem of some scavenger birds. *Membrane Computing. Proc. WMC10, Curtea de Argeş, 2009* (Gh. Păun et al., eds.), LNCS 5957, Springer, 2010, 182–195.
5. M. Cavaliere: Evolution-communication P systems. *Membrane Computing. Proc. WMC 2002, Curtea de Argeş* (Gh. Păun et al., eds.), LNCS 2597, Springer, Berlin, 2003, 134–145.
6. E. Csuhaj-Varjú: P automata. *Membrane Computing. Proc. WMC5, Milano, 2004* (G. Mauri et al., eds.), LNCS 3365, Springer, Berlin, 2005, 19–35.
7. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel-states. *Theoretical Computer Sci.*, 330, 1 (2005), 101–116.
8. J. Gruska: Descriptional complexity of context-free languages. *Proc. Symp. on Mathematical Foundations of Computer Science, MFCS*, High Tatras, 1973, 71–83.
9. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing.* Springer, Berlin, 1997.
10. M. Oswald: *P Automata.* PhD Thesis, TU Vienna, 2003.
11. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
12. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.
13. M.J. Pérez-Jiménez: A computational complexity theory in membrane computing. *Membrane Computing. Proc. WMC10, Curtea de Argeş, 2009* (Gh. Păun et al., eds.), LNCS 5957, Springer, 2010, 125–148.
14. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Introducing a space complexity measure for P systems. *Intern. J. Computers, Communications and Control*, 4, 3 (2009), 301–310.
15. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* 3 volumes, Springer, Berlin, 1998.
16. A.C. Yao: Some complexity questions related to distributed computing. *ACM Symposium on Theory of Computing*, 1979, 209–213.
17. The P Systems Website: `http://ppage.psystems.eu`.

Dynamics of Randomly Constructed Computational Systems

Miguel A. Peña[1], Pierluigi Frisco[2]

[1] Dpto. Inteligencia Artificial, Facultad de Informática
Universitad Politécnica de Madrid
Campus de Montegancedo, 28660 Madrid, Spain
m.pena@upm.es

[2] School of Mathematical and Computer Sciences
Heriot-Watt University, EH14 4AS Edinburgh, UK
P.Frisco@hw.ac.uk

Summary. We studied Petri nets with five places constructed in a pseudo-random way: their underlying net is composed of *join* and *fork*. We report initial results linking the dynamical properties of these systems to the topology of their underlying net.

The obtained results can be easily related to the computational power of some abstract models of computation.

1 Introduction

Recently [4, 5, 1, 7], several abstract models of computation operating with multisets of objects have been related to Petri nets. This study let to define new ways to prove the computational power of these abstract models of computations. Moreover, it also let a hierarchy of computational process to be defined. This hierarchy is based on *building blocks* (small Petri nets used to construct more complex Petri nets), the way the building blocks are combined, and the way the Petri net runs.

The results related to this hierarchy have been obtained using systems created *ad hoc*: the Petri nets were engineered in specific ways so to be able to generate specific languages. The languages generated by 'pseudo-random' Petri nets, remain to be investigated. Here 'random' refers to the fact that Petri nets are created composing building blocks in a random way, while 'pseudo' refers to the fact that some limitations to this randomness or to the way the Petri nets runs, are imposed. This direction of research was raised in [3] (suggestion for research 4).

In this paper we report our initial results on these investigations. The overall aims of this research is to be able to predict the behaviour of a computing system just looking at what has been called *topology of information flow* [6], that is, at the way the several parts of the system interact.

2 Basic Definitions

The model of Petri nets considered by us are known either as *elementary net systems* (*EN systems*), as *1-bounded place-transitions systems* (*1-bounded P/T systems*), or *safe Petri nets* [8].

An *elementary net system* (or *EN system*) is a tuple $N = (P, T, F, C_{in})$, where:

i) (P, T, F) is a *net*, that is:
  1. P and T are sets with $P \cap T = \emptyset$;
  2. $F \subseteq (P \times T) \cup (T \times P)$;
  3. for every $t \in T$ there exist $p, q \in P$ such that $(p, t), (t, q) \in F$;
  4. for every $t \in T$ and $p, q, \in P$, if $(p, t), (t, q) \in F$, then $p \neq q$;

ii) $C_{in} \subseteq P$ is the *initial configuration* (or *initial marking*).

Elements of P are called *places* (graphically represented with circles), elements of T are called *transitions* (graphically represented with rectangles). We use the common Petri net terminology and notation [8] with the exception of using the term *configuration* instead of *marking.*

We consider *maximal strategy* as running mode (i.e., the way transitions fire): in each configuration all transitions that can fire do so. Moreover, if in a configuration there is a conflict (two different transitions with a not empty intersection of input sets can fire), then all the transitions in the conflict fire. If after a firing a place should receive more than one token (from the firing of two different transitions), then only one token is assumed to be present in that place. This ensures that in every configuration places have at most one token and that the behaviour (sequence of configurations) of an EN system is deterministic.

This rather restrictive firing strategy (similar to the ones present in random Boolean networks [2]) has been mainly dictated by efficiency during these initial simulations. In section 5 we note that the firing strategy should be definitely changed in order to obtain results of a more general use.

We considered EN systems composed of only two building blocks: *join* and *fork* depicted in Figure 1.

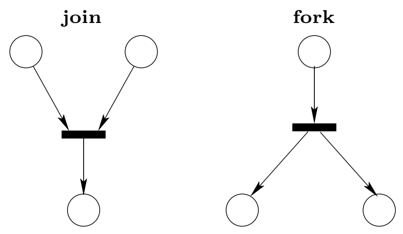


Fig. 1. Building blocks: *join* and *fork.*

Definition 1. *Let $x, y \in \{join, fork\}$ be building blocks and let $\bar{t}_x$ and $\hat{t}_y$ be the transitions present in x and y respectively.*

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

We say that a net is composed of *building blocks (it is* composed of x*) if it can be defined by building blocks (it is defined by* x*) sharing places but not transitions. So, for instance, to say that a net is* composed of joins *means that the only building blocks present in the net are* join*.*

3 The Simulator and Its Complexity

A computer program (in the following called *simulator*) able to create and run EN systems composed of *join* and *fork* has been written and it can be downloaded from `http://www.macs.hw.ac.uk/~pier/download.html`.

In the following j denotes the number of *join*, f denotes the number of *forks* and p denotes the number of places in a Petri net.

The maximum number of *join* (or *fork*) that can be present in a Petri net with p places is $\frac{p(p-1)(p-2)}{2}$. Moreover, when the Petri net is connected, $j + f \geq \frac{p}{2}$.

Given the number of places, the number of Petri nets with j *join* and f *forks* is

$$\frac{(\frac{p(p-1)(p-2)}{2})!}{j(\frac{p(p-1)(p-2)}{2} - j)!} * \frac{(\frac{p(p-1)(p-2)}{2})!}{f(\frac{p(p-1)(p-2)}{2} - f)!}$$

This number is definitely high even if one considers that it includes isomorphic nets. Due to this high number, we could only generate and run nets with 5 places. This means that the number of *join* and *fork* in these nets ranged from 1 to 30.

For each different triple of p, j and f, only 1% of the possible nets has been created and run for all its possible initial configurations.

The simulator created these nets in a random way.

The set of all possible configurations of such a net is called *configuration space*. The dynamics of an EN system is such that it will start from its initial configuration and it will reach an *attractor*. With *attractor* we define both a configuration from which no firing is possible or a set of configurations that are cyclically repeated. We name the configurations in the following way: With *isolated configuration* we refer to a configuration which is not reachable from any configuration and from which no transition is possible; with *final configuration* we refer to a configuration from which no transition is possible. Clearly, any isolated configuration is also a final configuration but a configuration can be final but not isolated.

The tests run on a computer with a single CPU of 2.4 GHz and with 1.5 GB of 800 MHz RAM. The simulation took 70 hours and the output files occupy 5 GB.

In Figure 2 a net and its configuration spaces are depicted. In this figure the configuration with no tokens is not shown (and in the following we do not consider this configuration). Each configuration in the configuration space is represented as a number in a circle. The number encodes the configuration of the EN system (as a conversion from binary to decimal). For instance, the encoding of configuration $\{0, 1, 0, 1, 1\}$ (nodes 0 and 2 have no token while the remaining nodes have 1 token) in the net depicted in Figure 2 is 11 (place 0 is the leftmost and place 4 is the rightmost).

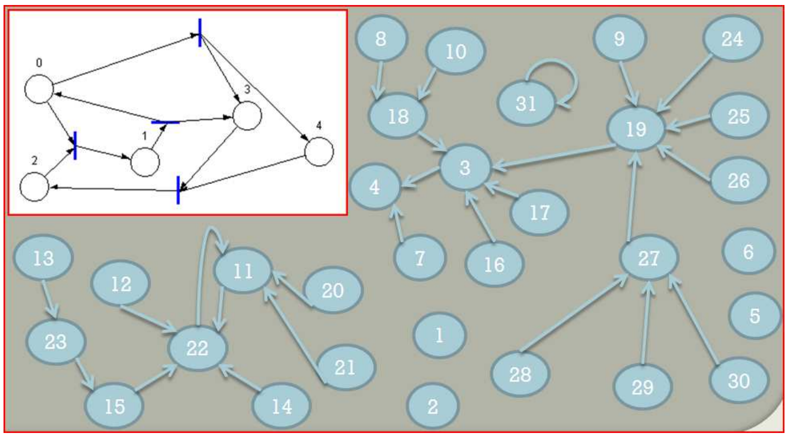


Fig. 2. A net and its configuration space

The attractors in Figure 2 are configurations 1, 2, 4, 5, 6, (11, 22) and 31, where (11, 22) define a *cycle* (i.e., an attractor with more than one configuration) in the configuration space. The configurations 1, 2, 5 and 6 are isolated (and final). Configuration 31 is not isolated as a transition (to itself) indeed is possible. Configurations 1, 2, 4, 5 and 6 are final.

4 Results

We addressed several questions during our study. The plotted answers to these questions and brief comments are present in the following. The tables used to generate the plots can be downloaded from
`http://www.macs.hw.ac.uk/~pier/cvPublications.html`

How does the probability to have at least one cycle in the configuration space depend on the number of join and fork?

We found that *join* and **fork** equally influence the presence of cycles in the configuration space. The plot in Figure 3 shows that if the number of *fork* is bigger than 9 or the number of *join* is bigger than 19, then it is certain that the configuration space contains at least one cycle.

How does the number of not-isolated configurations depend on the number of join and fork?

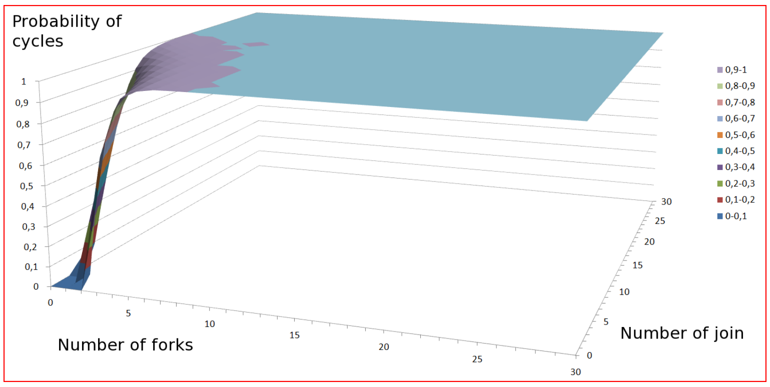


Fig. 3. Probability to have at least one cycle in the configuration space as a function of the number of *join* and *fork*

We found that the number of **fork** have a strong influence on the number of not-isolated configurations decreasing them to 0 with only 5 *forks* are present in the net. Also the increase of *join* tend to decrease the number of not-isolated configurations, but not with a marked effect as the number of *forks*. This is clearly shown by the plot in Figure 4.

Interestingly, the maximum number of not-isolated places is reached in nets with only 2 *joins*.

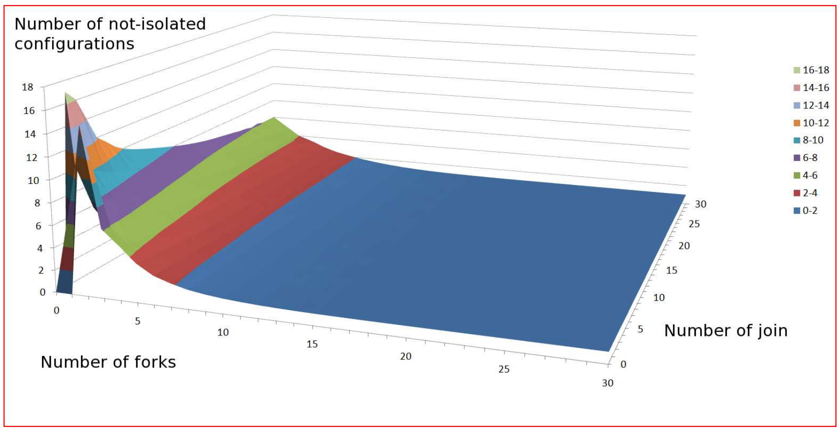


Fig. 4. Number of not-isolated configurations as a function of the number of *join* and *fork*

What is the minimum number of cycles present in the state space of Petri nets as a function of join and fork?

Also in this case the influence of *join* and *fork* is asymmetrical: a net can have up to 18 *join* and still no cycle is present in the state space of the Petri net. Differently, if the net has at least 11 *fork*, then the state space of the Petri net has at least 1 cycle. This is shown by the plot in Figure 5.

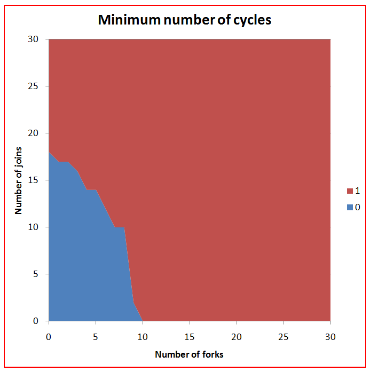


Fig. 5. Probability of the Petri net to end up in a cycle as a function of the number of *join* and *fork*

How many initial configurations will end up in a cycle depending on the number of join and fork?

The fact that the state space contains at least a cycle does not imply that all initial configurations will end up in in a cycle. It is confirmed that for high numbers of *join* and *fork* the Petri net will certainly enter a cycle. This is shown by the plot in Figure 6.

5 Final Remarks

A Petri net whose attractors are all final configurations can only generate or accept finite languages. Our study proved that the presence of only *join* or many *join* and a few *forks* let Petri nets have only final configurations as attractor. This result is rather immediate: a *join* consumes tokens, so if a net has only join, then sooner or later it will run out of tokens.

Thinks become more interesting when attractors with more than one place are present. In this case the set of languages generated or accepted is infinite. The kind of languages depends on the number of these attractors and their topology. We did not study this.

As said in Section 1, these results can be easily translated to formal models of computation operating with multisets of objects. Unfortunately, the firing strategy adopted by us, does not find a counterpart in any such model. This is, for instance, due to the fact that we allow one single token to be used in the firing of more than

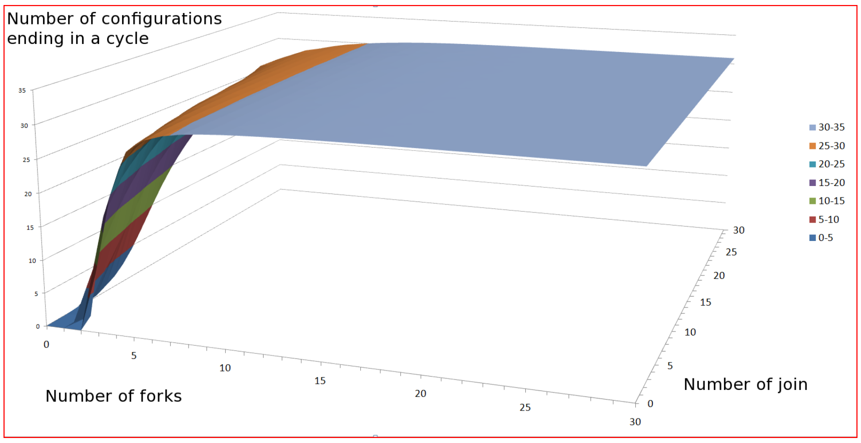


Fig. 6. Number of configurations that end up in a cycle as a function of the number of *join* and *fork*

one transition. The translation of this feature in, for instance, P systems, means that one single occurrence of an object can be used in the same configuration by different rules.

For this reason one of our future direction of research will be to implement firing strategies closer to the operational modes of existing formal models of computation.

Acknowledgment

The work of Miguel A. Peña was supported by a grant for short stays in Spain and abroad for the beneficiaries of official pre-doctoral programs for training researchers, year 2009, from the Technical University of Madrid.

References

1. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
2. B. Drossel. Random boolean networks. http://arxiv.org/pdf/0706.3351.
3. P. Frisco. P systems and topology: some suggestions for research. Seventh Brainstorming Week on Membrane Computing, 2009, pp. 123-132, Universidad de Sevilla, Technical report num. 1/2009, volume 1.
4. P. Frisco. P systems, Petri nets, and Program machines. In R. Freund, G. Lojka, M. Oswald, and G. Păun, editors, *Membrane Computing. 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*, volume 3850 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, Berlin, Heidelberg, New York, 2006.

5. P. Frisco. *Computing with Cells. Advances in Membrane Computing.* Oxford University Press, 2009.
6. P. Frisco. Conformon P systems and topology of information flow. In G. Păun, editor, *Membrane Computing. 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009, Revised Selected and Invited Papers*, volume 5957 of *Lecture Notes in Computer Science*, pages 30–53. Springer-Verlag, Berlin, Heidelberg, New York, 2009.
7. Z. Qi, J. You, and H. Mao. P systems and petri nets. volume 2933 of *Lecture Notes in Computer Science*, pages 286–303. Springer-Verlag, Berlin, Heidelberg, New York, 2004.
8. G. Rozenberg and J. Engelfriet. *Elementary net systems*, volume 1491 of *Lecture Notes in Computer Science*, pages 12–121. Springer-Verlag, Berlin, Heidelberg, New York, 1998.

Complete Problems for a Variant of P Systems with Active Membranes

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
`{porreca,leporati,mauri,zandron}@disco.unimib.it`

Summary. We identify a family of decision problems that are hard for some complexity classes defined in terms of P systems with active membranes working in polynomial time. Furthermore, we prove the *completeness* of these problems in the case where the systems are equipped with a form of priority that linearly orders their rules. Finally, we highlight some possible connections with open problems related to the computational complexity of P systems with active membranes.

1 Introduction

Membrane systems, usually called *P systems*, are computing devices inspired by the internal working of biological cells [6]. The main feature of P systems is a structure of membranes dividing the space into regions, inside which multisets of objects describe the molecular environment. A set of rules describe how the molecules (and often the membranes themselves) evolve during the computation; usually, the rules are applied in a maximally parallel way, i.e., each component of the P system *must* be subject to a rule during each computation step, if a suitable rule exists. When multiple rules may be applied to an object or membrane, one of them is nondeterministically chosen. The computation, starting from an initial configuration, proceeds until no further rule can be applied. For an introduction on membrane computing we refer the reader to [8, 9], and for the latest information to the P Systems Webpage [16], where an extensive bibliography on the topic can be found.

Families of P systems can be used as language recognizers, by associating with each input string (or to each input length) a P system; this association is subject to a uniformity condition (i.e., it must be computed by a Turing machine operating in polynomial time). The constructed P systems can then accept or reject, thus deciding the membership of strings to the language. The computational complexity of recognizer P systems with *active membranes* [7], where the membranes themselves play an important role during the computation, has been subject to extensive investigation, due to their ability of solving **NP**-complete and even **PSPACE**-complete problems [11] in polynomial time: this efficiency is due to the possibility of creating in polynomial time an exponential number of membranes, which then evolve in parallel using *membrane division* (a process found in nature, e.g., in cell mitosis).

In this paper we investigate the existence of complete problems for complexity classes defined in terms of P systems with active membranes. Some classes are known to possess them, simply because they coincide with classes defined in terms of Turing machines and inherit their complete problems: for instance, polynomial-time P systems with active membranes without membrane division characterize **P** [15], while they caracterize **PSPACE** if all kinds of rule are available [12]. Our approach is, however, more general; we exhibit problems (inspired by analogous ones for Turing machines) that are hard for *every* polynomial-time complexity class defined in terms of P systems with active membranes, independently of what rules are available, and complete for several of them if a restricted form of priority among rules is used.

The remainder of this paper is organized as follows. In Section 2 we recall the notion of recognizer P systems with active membranes and how to measure their time complexity. Section 3 describes the bounded acceptance problem; in particular, the variant for nondeterministic Turing machines is considered and its **NP**-completeness is proved. In Section 4 we introduce the bounded acceptance problem for P systems and prove its hardness for all polynomial-time complexity classes; we also prove that the problem is complete if we add a linear ordering on the rules of the P systems.

2 Definitions

We begin by recalling the definition of P systems with active membranes.

Definition 1. *A* P system with active membranes *of the initial degree* $m \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_m, R)$$

where:

- Γ *is a finite alphabet of symbols, also called* objects*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted unordered tree) consisting of* m *membranes enumerated by* $1, \ldots, m$*; furthermore, each membrane is labeled by an element of* Λ*, not necessarily in a one-to-one way;*
- $w_1, \ldots, w_m$ *are strings over* Γ*, describing the multisets of objects placed in the* m *initial regions of* μ*;*
- R *is a finite set of rules.*

Each membrane possesses a further attribute, named *polarization* or *electrical charge*, which is either neutral (represented by 0), positive (+) or negative (−) and it is assumed to be initially neutral.

The rules are of the following kinds:

- *Object evolution rules*, of the form $[a \to w]_h^\alpha$
  They can be applied inside a membrane labeled by h, having polarization α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by every object in w).
- *Send-in communication rules*, of the form $a\,[\;]_h^\alpha \to [b]_h^\beta$
  They can be applied to a membrane labeled by h, having polarization α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the polarization of h is changed to β.
- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\;]_h^\beta\, b$
  They can be applied to a membrane labeled by h, having polarization α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the polarization of h is changed to β.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labeled by h, having polarization α and containing an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b.
- *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$
  They can be applied to a membrane labeled by h, having polarization α, containing an occurrence of the object a but having no other membrane inside; the membrane is divided into two membranes having label h and polarizations β and γ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes.
- *Non-elementary division rules*, of the form

$$[[\;]_{h_1}^+ \cdots [\;]_{h_k}^+ [\;]_{h_{k+1}}^- \cdots [\;]_{h_n}^-]_h^\alpha \to [[\;]_{h_1}^\delta \cdots [\;]_{h_k}^\delta]_h^\beta [[\;]_{h_{k+1}}^\varepsilon \cdots [\;]_{h_n}^\varepsilon]_h^\gamma$$

  They can be applied to a membrane labeled by h, having polarization α, containing the positively charged membranes $h_1, \ldots, h_k$, the negatively charged membranes $h_{k+1}, \ldots, h_n$, and possibly some neutral membranes. The membrane h is divided into two copies having polarization β and γ, respectively; the positive children are placed inside the former, their polarizations changed to δ, while the negative ones are placed inside the latter, their polarizations changed to ε. Any neutral membrane inside h is duplicated and placed inside both copies.

A *configuration* in a P system with active membranes is described by its current membrane structure, together with its polarizations and the multisets of objects contained in its regions. The *initial configuration* is given by μ, all membranes having polarization 0 and the initial contents of the membranes being $w_1, \ldots, w_m$. A *computation step* changes the current configuration according to the following principles:

- Each object and each membrane can be subject to only one rule during a computation step.
- The rules are applied in a *maximally parallel way*: each object which appears on the left-hand side of applicable evolution, communication, dissolution or elementary division rules must be subject to exactly one of them; the same holds for each membrane which can be involved in a communication, dissolution or division rule. The only objects and membranes which remain unchanged are those associated with no rule, or with unapplicable rules.
- When more than one rule can be applied to an object or membrane, the actual rule to be applied is chosen nondeterministically; hence, in general, multiple configurations can be reached from the current one.
- When dissolution or division rules are applied to a membrane, the multiset of objects to be released outside or copied is the one resulting *after* all evolution rules have been applied.
- The skin membrane cannot be divided, nor it can be dissolved. Furthermore, every object which is sent out from the skin membrane cannot be brought in again.

A *halting computation* $\boldsymbol{\mathcal{C}}$ of a P system Π is a finite sequence of configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration of Π, every $\mathcal{C}_{i+1}$ can be reached from $\mathcal{C}_i$ according to the principles just described, and no further configuration can be reached from $\mathcal{C}_k$ (i.e., no rule can be applied). P systems might also perform *non-halting* computations; in this case, we have infinite sequences $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ of successive configurations.

We can use families of P systems with active membranes as language recognizers, thus allowing us to solve decision problems.

Definition 2. *A* recognizer P system with active membranes Π *has an alphabet containing two distinguished objects yes and no, used to signal acceptance and rejection respectively; every computation of Π is halting and exactly one object among yes, no is sent out from the skin membrane during each computation.*

In what follows we will only consider *confluent* recognizer P systems with active membranes, in which all computations starting from the initial configuration agree on the result.

Definition 3. *Let $L \subseteq \Sigma^\star$ be a language and let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems. We say that $\boldsymbol{\Pi}$* decides *L, in symbols $L(\boldsymbol{\Pi}) = L$, when for each $x \in \Sigma^\star$, the result of Π_x is acceptance iff $x \in L$.*

Usually some uniformity condition, inspired by those applied to families of Boolean circuits, is imposed on families of P systems. Two different notions of uniformity have been considered in the literature; they are defined as follows.

Definition 4. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* semi-uniform *when the mapping* $x \mapsto \Pi_x$ *can be computed in polynomial time, with respect to* $|x|$*, by a deterministic Turing machine.*

Definition 5. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* uniform *when there exist two polynomial-time Turing machines* M_1 *and* M_2 *such that, for each* $n \in \mathbb{N}$ *and each* $x \in \Sigma^n$

- M_1*, on input* 1^n *(the unary representation of the length of* x*), outputs the description of a P system* Π_n *with a distinguished input membrane;*
- M_2*, on input* x*, outputs a multiset* w_x *(an encoding of* x*);*
- Π_x *is* Π_n *with* w_x *added to the multiset located inside its input membrane.*

In other words, the P system Π_x *associated with string* x *consists of two parts; one of them,* Π_n*, is common for all strings of length* $|x| = n$ *(in particular, the membrane structure and the set of rules fall into this category), and the other (the input multiset* w_x *for* Π_n*) is specific to* x*. The two parts are constructed independently and, only as the last step,* w_x *is inserted in* Π_n*.*

Time complexity classes for P systems [10] are defined as usual, by restricting the amount of time available for deciding a language. By $\mathbf{PMC}_{\mathcal{D}}$ (resp., $\mathbf{PMC}^\star_{\mathcal{D}}$) we denote the class of languages which can be decided by uniform (resp., semi-uniform) families $\boldsymbol{\Pi}$ of confluent P systems of class $\mathcal{D}$ (e.g., $\mathcal{AM}$ denotes the class of P systems with active membranes) where each computation of $\Pi_x \in \boldsymbol{\Pi}$ halts in polynomial time with respect to $|x|$.

3 The bounded acceptance problem for Turing machines

A classic decision problem related to each class of automata is the *acceptance* or *prediction problem*: given an automaton A and a string x over its input alphabet, is it the case that A accepts x? The difficulty of this problem varies according to the class of automata: for instance, if we consider finite automata it is in **P**, while it is **PSPACE**-complete for linear bounded automata [3, p. 265]; one of the first and most important results of computability theory asserts that the problem is undecidable for Turing machines [13].

Even when the problem is not solvable, one can often devise an easier version by limiting the amount of resources allocated. Consider the variant defined as follows.

Definition 6. *The* bounded acceptance problem $\mathrm{BAP}_{\mathrm{NTM}}$ *for nondeterministic Turing machines is the set of triples* $(N, x, 1^t)$ *where*

- N *is a "reasonable" encoding [2] of a nondeterministic Turing machine;*
- x *is a string over the input alphabet of* N*;*
- 1^t *is the unary encoding of a natural number* t*;*

such that N *accepts the string* x *within* t *computation steps.*

$\mathrm{BAP}_{\mathrm{NTM}}$ is an interesting problem because it is very easy to prove its **NP**-completeness.

Proposition 1. $\mathrm{BAP}_{\mathrm{NTM}} \in \mathbf{NP}$.

Proof (a variant of the proof of Theorem 2.9 in [1]). A nondeterministic Turing machine N', given $(N, x, 1^t)$ as input, can easily perform a step-by-step simulation of a computation of N on x with a polynomial slowdown, making a nondeterministic choice every time N does; simultaneously, on another tape, N' counts the number of simulated computation steps. If N accepts before the counter reaches $t + 1$, then N' halts and accepts; if, on the contrary, the counter reaches $t + 1$ without N having accepted, then N' halts and reject. Since the input t is given in unary notation, the whole simulation runs in polynomial time, and N' has an accepting computation on $(N, x, 1^t)$ iff N has an accepting computation on x. □

Proposition 2. $\mathrm{BAP}_{\mathrm{NTM}}$ *is* **NP**-*hard.*

Proof. Let $L \in \mathbf{NP}$ be a language. Then, there exists a nondeterministic Turing machine N deciding L in polynomial time $p(n)$.

Let $R\colon \Sigma^\star \to \Sigma^\star$ be defined by $R(x) = (N, x, 1^{p(|x|)})$; the function R can be computed by a *deterministic* Turing machine operating in polynomial time, since N does not depend on x (hence it can be output in constant time) and $1^{p(|x|)}$ can be easily computed in polynomial time given a unary encoding of $|x|$ as input (which can be easily obtained from x itself).

Now let $x \in \Sigma^\star$. Clearly $x \in L$ iff N accepts x; since N runs in $p(n)$ steps, this is equivalent to saying that $R(x) = (N, x, 1^{p(|x|)}) \in \mathrm{BAP}_{\mathrm{NTM}}$: thus, R is a polynomial-time reduction of L to $\mathrm{BAP}_{\mathrm{NTM}}$. □

4 The bounded acceptance problem for P systems

In this section we discuss a variant of the bounded acceptance problem in the setting of P systems with active membranes.

Definition 7. *Let $\mathcal{D}$ be any subclass of P systems with active membranes; the* bounded acceptance problems $\mathrm{BAP}_{\mathcal{D}}$ *and* $\mathrm{BAP}^\star_{\mathcal{D}}$ *for uniform and semi-uniform families of P systems of class $\mathcal{D}$ are, respectively, the set of triples $(\Pi, w, 1^t)$ and the set of pairs $(\Pi, 1^t)$ where*

- *Π is a P system in the class $\mathcal{D}$;*
- *w is a multiset over the alphabet of Π;*
- *1^t is the unary encoding of a natural number t;*

such that Π accepts within t steps when the multiset w is placed inside its input membrane (resp., Π accepts within t steps).

It is easy to show that every problem in $\mathbf{PMC}_{\mathcal{D}}$ can be reduced to $\mathrm{BAP}_{\mathcal{D}}$ (and every problem in $\mathbf{PMC}^\star_{\mathcal{D}}$ to $\mathrm{BAP}^\star_{\mathcal{D}}$) by modifying the proof of Proposition 2.

Lemma 1. $\mathrm{BAP}_{\mathcal{D}}$ *is* $\mathbf{PMC}_{\mathcal{D}}$*-hard, and* $\mathrm{BAP}^{\star}_{\mathcal{D}}$ *is* $\mathbf{PMC}^{\star}_{\mathcal{D}}$*-hard.*

Proof. Let $L \in \mathbf{PMC}_{\mathcal{D}}$ be a language. Then, there exists a uniform family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ deciding L in polynomial time $p(n)$; the family $\boldsymbol{\Pi}$ is constructed by polynomial-time Turing machines M_1 (for the portion that only depends on the length of the input) and M_2 (for the input multiset, which depends on the whole string).

Let $R\colon \Sigma^\star \to \Sigma^\star$ be defined by $R(x) = \big(M_1(|x|), M_2(x), 1^{p(|x|)}\big)$; clearly, the function R can be computed in polynomial time by a deterministic Turing machine. For each $x \in \Sigma^\star$ we have, by hypothesis, $x \in L$ iff the P system Π_x, constructed by combining $M_1(|x|)$ with the multiset $M_2(x)$, accepts within $p(|x|)$ steps; in other words, $x \in L$ iff $R(x) \in \mathrm{BAP}_{\mathcal{D}}$. But then R is a polynomial-time reduction from L to $\mathrm{BAP}_{\mathcal{D}}$.

The proof is completely analogous in the semi-uniform case: the only difference is that we have no input multiset. $\square$

Unfortunately, proving that the BAP for standard P systems with active membranes belongs to $\mathbf{PMC}_{\mathcal{D}}$ or $\mathbf{PMC}^{\star}_{\mathcal{D}}$ (if this is indeed the case) seems to be much more difficult than proving the same result for Turing machines. The reason is that it is very easy to equip a Turing machine with a clock that halts the machine after a certain number of steps: there are few “moving parts” to stop, namely the tape heads, and the transition function controls them all simultaneously. On the other hand, in a P system the global state is distributed, and each rule only affects a small part of it; the computation is also, in general, nondeterministic. It is then very difficult to stop every region of the P system, at least in an efficient way (i.e., with a polynomial slowdown): the intuitive idea of halting the system by dissolving every membrane when a certain period of time has passed also fails, since the dissolution rule may enter into a conflict with the original rules, and hence might never be applied.

4.1 Completeness of BAP in a special case

By adding a limited form of priority among rules to P systems with active membranes (both with and without polarizations), we are able to find a class $\mathcal{D}$ such that a variant of $\mathrm{BAP}_{\mathcal{D}}$ (resp., $\mathrm{BAP}^{\star}_{\mathcal{D}}$) is complete for $\mathbf{PMC}_{\mathcal{D}}$ (resp., $\mathbf{PMC}^{\star}_{\mathcal{D}}$).

Definition 8. *A* P system with active membranes *(with or without polarizations)* and linear priority

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_m, R, \leq)$$

is a P system $(\Gamma, \Lambda, \mu, w_1, \ldots, w_m, R)$ with active membranes equipped with a linear (i.e., total) order $\leq$ over R, which is used as follows: whenever a nondeterministic choice between two rules $r_1 < r_2$ has to be made during the computation, the rule which is actually applied is r_2.

Confluent families of P systems with active membranes and polarizations using only elementary membrane division are powerful enough to solve **NP**-complete problems in polynomial time [15]. Due to confluence, we know that any possible computation (i.e., any sequence of nondeterministic, maximally paralles choices of rules) yields the correct result: in particular, the computation obtained by linearly ordering the set of rules in an arbitrary way. This proves that introducing priorities does not affect the ability of solving **NP**-complete problems in polynomial time. On the other hand, one can simulate polynomial-time P systems with active membranes in polynomial space, and this simulation can be easily extended to include priorities without increasing the space requirements[12]. In other words, the class of problems decided in polynomial time by P systems with active membranes (with polarizations) and linear priority is located between **NP** and **PSPACE**.

Proposition 3. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{D}} \subseteq \mathbf{PMC}^{\star}_{\mathcal{D}} \subseteq \mathbf{PSPACE}$. □

The existence of complete problems for $\mathbf{PMC}_{\mathcal{D}}$ and $\mathbf{PMC}^{\star}_{\mathcal{D}}$ not only provides us with a possible way of proving $\mathbf{PMC}_{\mathcal{D}} \subseteq \mathbf{NP}$ and $\mathbf{PMC}^{\star}_{\mathcal{D}} \subseteq \mathbf{NP}$ by solving one of these problems via a polynomial-time nondeterministic Turing machines, but it also gives us another hint on the nature of this complexity class. It is a common assumption [5] that the the cumulative polynomial hierarchy **PH** (i.e., the union of all levels of the hierarchy), which is also located between **NP** and **PSPACE**, consists of infinitely many distinct levels. As a consequence, **PH** is conjectured not to have complete problems, since their existence would imply the *collapse* of the hierarchy, i.e., only finitely many levels would exist: we can then conjecture that $\mathbf{PMC}_{\mathcal{D}}$ and $\mathbf{PMC}^{\star}_{\mathcal{D}}$ differ from **PH** (or, for that matter, from any other class lacking complete problems).

On the other hand, if we consider *polarizationless* P systems with active membranes and elementary division only, we run into the so-called *P conjecture* [14, 4]: these systems are conjectured to solve only **P** problems in polynomial time, but a proof of this statement is still missing. We are able to exhibit a complete problem when linear priority among rules is assumed: solving it, or solving any new complete problem which could emerge in the future, via a polynomial-time deterministic Turing machine (if this is possible at all) might provide some insight for the P conjecture.

The bounded acceptance problem for P systems with active membranes and linear priority is hard for $\mathbf{PMC}_{\mathcal{D}}$ and $\mathbf{PMC}^{\star}_{\mathcal{D}}$ as described above, and its completeness can be proved as follows.

Theorem 1. *Let $\mathcal{D}$ be any subclass of P systems with active membranes (with or without polarizations) and linear priority, using at least evolution, communication and dissolution rules. Then* $\mathrm{BAP}_{\mathcal{D}}$ *and* $\mathrm{BAP}^{\star}_{\mathcal{D}}$ *are complete for* $\mathbf{PMC}_{\mathcal{D}}$ *and* $\mathbf{PMC}^{\star}_{\mathcal{D}}$ *respectively.*

Proof. We only have to prove membership in $\mathbf{PMC}_{\mathcal{D}}$ (we focus on the uniform version here, as the argument for the semi-uniform one is just a simplification of it). Given $(\Pi, w, 1^t)$, a polynomial-time Turing machine M_1 can construct a P system Π' which is identical to Π except for the following differences:

- The membrane structure is enclosed by two further membranes with new labels h_1 and h_0 (h_0 becomes the new skin membrane).
- The objects *yes* and *no* of Π are renamed as the two new symbols yes' and no' (they are also renamed in all rules involving them).
- Each of the original membranes of Π contains a timer object z_0, that evolves to the new objects $z_1, \dots, z_t$ according to the rules

$$[z_0 \to z_1]_h^\alpha \qquad [z_1 \to z_2]_h^\alpha \qquad \cdots \qquad [z_{t-1} \to z_t]_h^\alpha \qquad [z_t]_h^\alpha \to \#$$

  for all $\alpha \in \{+, 0, -\}$ and for each original label h. These objects count up to t and then dissolve the membrane producing a new "junk" object. In the polarizationless case, the charges are simply omitted.
- The new membrane h_1 also contains a timer object z'_0 with the rules

$$[z'_0 \to z'_1]_{h_1}^\alpha \qquad [z'_1 \to z'_2]_{h_1}^\alpha \qquad \cdots \qquad [z'_t \to z'_{t+1}]_{h_1}^\alpha \qquad [z'_{t+1}]_{h_1}^\alpha \to no$$

  The goal of these objects is to count up to $t+1$ and then dissolve h_1, producing the object *no*. The objects z'_i are also assumed to be different from any object in Π.
- The new membranes h_1 and h_0 are also involved in the following rules:

$$\begin{array}{ll} [yes']_{h_1} \to yes & \qquad [no']_{h_1} \to no \\ [yes]_{h_0} \to [\,]_{h_0}\ yes & \qquad [no]_{h_0} \to [\,]_{h_0}\ no \end{array}$$

- The priority among the original rules is left untouched, but all the new rules have a higher priority than them; the precise ordering among the new rules is arbitrary.

The Turing machine M_2 constructing the input multiset simply outputs the multiset w. Thus, we obtain a uniform family $\boldsymbol{\Pi}$ of P systems.

If the original P system Π sends out its output from its skin membrane within t steps, then either yes' or no' appears in membrane h_1 of Π'. In such a case, *yes* (resp., *no*) is first sent out to h_0 (by dissolution, hence interrupting the timer z') and then to the environment as the result of the computation, which in this case is clearly the same as Π. If, however, Π computes for more than t steps, then its membranes are all simultaneously dissolved in Π' in step $t+1$ (recall that the new rules have highest priority) and membrane h_1 produces a *no* object that becomes the result of the computation. Thus, the family $\boldsymbol{\Pi}$ solves $\mathrm{BAP}_{\mathcal{D}}$, and it does so in $O(t)$ steps, which is polynomial time with respect to the input size. □

5 Conclusions

We have shown that the bounded acceptance problem, that is, determining whether a given P systems accepts its input within a certain number of steps, is complete for the class of decision problems that can be solved in polynomial time by certain families of P systems. The P systems covered by the proof are recognizer P systems with active membranes, with or without polarizations, using at least evolution, communication and dissolution rules; additionally, it is required that a linear priority relation is defined on the rules of the system. The result holds both in the uniform and semi-uniform cases.

The classes of decision problems $\mathbf{PMC}_{\mathcal{D}}$ and $\mathbf{PMC}^{\star}_{\mathcal{D}}$ defined as above are easily shown to be located between **NP** and **PSPACE**: hence, the existence of decision problems for these classes might provide an useful tool in finding a tighter upper bound, as well as in differentiating them from classes, such as the cumulative polynomial hierarchy **PH**, that do not (apparently) possess complete problems.

An outstanding open question concerns the existence of complete problems for complexity classes defined in terms of *standard* (i.e., without priority) polynomial-time P systems with active membranes, particularly those without non-elementary division rules, as they still lack a characterization in terms of Turing machines; investigating this question in the polarizationless case might shed new light on the *P conjecture*.

Acknowledgements

This work was partially supported by the Italian project FIAR 2008 “Modelli di calcolo naturale e applicazioni alla systems biology”.

References

1. S. Arora, B. Barak: *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.
2. J.L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity I.* Springer, 1995.
3. M.R. Garey, D.S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co., 1979.
4. G. Mauri, M.J. Pérez-Jiménez, C. Zandron: On a Păun's conjecture in membrane systems. *Second International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2007* (J. Mira, J.R. Álvarez, eds.), volume 4527 of *Lecture Notes in Computer Science.* Springer, 2007, 180–192.
5. C.H. Papadimitriou: *Computational Complexity.* Addison-Wesley, 1993.
6. G. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 1, 61 (2000), 108–143.
7. G. Păun: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 75–90.
8. G. Păun, G. Rozenberg: A guide to membrane computing. *Theoretical Computer Science*, 287 (2002), 73–100.
9. G. Păun, G. Rozenberg, A. Salomaa (eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
10. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2, 3 (2003), 265–285.

11. P. Sosík: The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2, 3 (2003), 287–298.
12. P. Sosík, A. Rodríguez-Patón: Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73, 1 (2007), 137–152.
13. A.M. Turing: On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42 (1936), 230–265.
14. D. Woods, N. Murphy, M.J. Pérez-Jiménez, A. Riscos-Núñez: Membrane dissolution and division in P. *Unconventional Computation, 8th International Conference, UC 2009* (C.S. Calude, J.F. Costa, N. Dershowitz, E. Freire, G. Rozenberg, eds.), volume 5715 of *Lecture Notes in Computer Science*. Springer, 2009, 262–276.
15. C. Zandron, C. Ferretti, G. Mauri: Solving NP-complete problems using P systems with active membranes. *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Brussel, Belgium, 13–16 December 2000, Springer, 2001, 289–301.
16. *The P Systems Webpage*, `http://ppage.psystems.eu`.

First Steps Towards Linking Membrane Depth and the Polynomial Hierarchy

Antonio E. Porreca[1], Niall Murphy[2]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca, Italy
porreca@disco.unimib.it
[2] Department of Computer Science
National University of Ireland Maynooth, Ireland
nmurphy@cs.nuim.ie

Summary. In this paper we take the first steps in studying possible connections between non-elementary division with limited membrane depth and the levels of the Polynomial Hierarchy. We present a uniform family with a membrane structure of depth $d + 1$ that solves a problem complete for level d of the Polynomial Hierarchy.

1 Introduction

Active membrane systems without charges are an extremely interesting group of models to study from the computational complexity point of view. Forbidding the use of a single rule type yields dramatic differences in computing power of these models. For example, it is known that systems with strong non-elementary division characterise PSPACE [1, 14], but when dissolution is forbidden these systems can solve at most problems in NL in the AC^0-semi-uniform case [7], and at most AC^0 in the AC^0-uniform case [8]. Since $\mathsf{AC}^0 \subsetneq \mathsf{NL} \subsetneq \mathsf{PSPACE}$ it seems these rules somehow capture different aspects of computation.

In this report we present our first step towards a better understanding of the difference between P and PSPACE in terms of membrane systems. We suspect that the depth of a membrane system combined with non-elementary division is the key to this difference. Non-elementary division an operation where a membrane divides and all child membranes (and their child membranes etc.) get copied. There are two varieties of non-elementary division, “strong” which is triggered by membranes, and “weak” which is triggered by objects. (The labels “weak” and “strong” have nothing to do with the power of these rules.) Elementary division is where division is only permitted on membranes that do not have child membranes, and can be thought of as non-elementary division on structure of depth of 0.

- Systems with *strong* non-elementary division and polynomial membrane depth are known to characterise PSPACE [1, 14].

- Systems with *weak* non-elementary division and polynomial depth can solve at least all of NP ∪ coNP [3].
- Systems with elementary division (non-elementary division on depth 0) are believed to characterise P (see [6, 16] for some partial results, this is an open problem known as the P-conjecture [5, 12]).

This has lead us to an intriguing hypothesis: that by using non-elementary division rules and by limiting the depth of the membrane structure we can characterise each level of the polynomial hierarchy from P to PSPACE. If this hypothesis is correct it will help us understand how membrane division contributes in the jump from P to PSPACE and will help resolve the P-conjecture.

The idea that increasing the depth of the membrane structure also increases the computing power of the systems is also consistent with another recent result. Porreca et al. [13] show that (if no time limit is imposed) increasing the depth of active membrane systems using only communication and strong non-elementary division rules permits the systems to solve exponentially harder problems.

This report presents our first steps to proving a link between non-elementary division for a specific membrane depth and the polynomial hierarchy. We show that logspace uniform families of membrane system with a structure of depth $d+1$ can solve problems complete for the d^{th} level of the polynomial hierarchy. In other words, adding a further level of depth gives us the power of an oracle for the previous level of the hierarchy.

In future work we hope to find a corresponding upper-bound where a Turing machine with d alternations can simulate a membrane system with non-elementary division and depth $d+1$.

2 Definitions for Membrane Systems

In this section we define membrane systems and some complexity classes, these definitions are based on those from [4, 11, 9, 10, 14]. The set of all multisets over a set A is denoted $\mathrm{MS}(A)$.

2.1 Active membrane systems

Active membrane systems are a class of membrane systems with membrane division rules. In this paper we use division rules that can act on elementary membranes, which are membranes that do not contain other membranes (i.e. leaves in the membrane structure), or non-elementary membranes, membranes that do contain other membranes.

Definition 1. *An active membrane system without charges is a 6-tuple $\Pi = (O, \mu, M, H, L, R)$ where,*

1. *O is the alphabet of objects;*

2. *$\mu = (V_\mu, E_\mu)$ is a tree representing the membrane structure, where $V_\mu \subseteq \mathbb{N}$ and $E_\mu \subsetneq V_\mu \times V_\mu$;*
3. *$M : V_\mu \rightarrow \mathrm{MS}(O)$ maps membranes to their multisets;*
4. *H is the finite set of membrane labels;*
5. *$L : V_\mu \rightarrow H$ maps membranes to their labels;*
6. *R is a finite set of developmental rules of the following types (where $a, b, c \in O$ and $u \in \mathrm{MS}(O)$, $h \in H$):*
   (a) *$[\, a \rightarrow u\,]_h$ (object evolution),*
   (b) *$a\, [\ ]_h \rightarrow [\, b\,]_h$ (communication in),*
   (c) *$[\, a\,]_h \rightarrow [\ ]_h\ b$ (communication out),*
   (d) *$[\, a\,]_h \rightarrow b$ (membrane dissolution),*
   ($\mathrm{e_w}$) *$[\, a\,]_h \rightarrow [\, b\,]_h\ [\, c\,]_h$, (weak non-elementary membrane division).*

The vertices V_μ of the membrane structure tree μ are the individual membranes of the system. The parent of all membranes in the system (the root vertex in μ) is called the "skin" and has label $0 \in H$. A *configuration* $\mathcal{C}$ of a membrane system is a tuple (μ, M, L) whose elements are defined in Definition 1. A *permissible encoding* of a membrane system $\langle \Pi \rangle$, or a configuration $\langle \mathcal{C} \rangle$, encodes all multisets in a unary manner. For example, a multiset must be specified in the format $[\, a, a, a, b, b\,]$, rather than a^3b^2, in order to ensure that at most a polynomial number of objects are initially encoded in a system.

The rules in the set R are applied to a configuration according to the following principles:

- All the rules are applied in a *maximally parallel manner*. In each timestep, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do so.
- If a membrane labelled h is divided by a rule of type (e) and there are objects in this membrane which evolve via rules of type (a), then we assume that first the evolution (a) rules are used, and then the division (e) rules. This process takes only one step.
- The rules associated with membranes labelled with h are used for membranes with that label. In each timestep, a membrane can be the subject of only one rule of types (b)–($\mathrm{e_w}$).

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

Definition 2. *A* recogniser membrane system *is a membrane system Π such that:*

1. *all computations halt,*
2. `yes`, `no` $\in O$,

3. *the object* `yes` *or object* `no` *(but not both) appear in the multiset of the membrane with label* 0 *(the skin),*
4. *and this happens only in the halting configuration.*

2.2 Complexity classes

A problem is a set $X = \{x_1, x_2, \ldots\} \subseteq \Sigma^*$ and its complement is $\overline{X} = \Sigma^* - X$ where Σ is some finite alphabet. We say that a *family* $\mathbf{\Pi}$ of membrane systems recognises a problem if for each $x \in \Sigma^*$ there is some $\Pi \in \mathbf{\Pi}$ that decides if $x \in X$. We denote by $|x| = n$ the length of any instance $x \in \Sigma^*$. Throughout this paper, AC^0 circuits are $\mathsf{DLOGTIME}$-uniform, polynomial sized (in input length n), constant depth, circuits with AND, OR and NOT gates, and unbounded fan-in [2]. FP, FL, and FAC^0 are the classes of functions that are respectively computable by deterministic Turing Machines in polynomial time, by deterministic Turing machines using logarithmic space, and by $\mathsf{DLOGTIME}$-uniform polynomial-sized alternating circuits with unbounded fan-in and constant depth.

Definition 3. *Let $\mathcal{R}$ be a class of recogniser membrane systems and let $t : \mathbb{N} \to \mathbb{N}$ be a total function. Let* E *and* F *be classes of functions. The class of* problems solved by a (E, F)-uniform family of membrane systems *of type $\mathcal{R}$ in time t, denoted* (E, F)–$\mathsf{MC}_{\mathcal{R}}(t)$*, contains all problems X such that:*

- *There exists an* F-uniform *family of membrane systems,* $\mathbf{\Pi} = \{\Pi_1, \Pi_2, \ldots\}$ *of type $\mathcal{R}$: that is, there exists a function $f \in \mathsf{F}$, $f : \{1\}^* \to \mathbf{\Pi}$ such that $f(1^n) = \Pi_n$, where $|x| = n$.*
- *There exists an input encoding function $e \in \mathsf{E}$, $e : X \cup \overline{X} \to \mathrm{MS}(I)$ such that $e(x)$ is the input multiset, which is placed in a specific input membrane of Π_n, where $|x| = n$ and $I \subsetneq O$ is the set of input objects.*
- $\mathbf{\Pi}$ *is t-efficient: Π_n always halts in at most $t(n)$ steps.*
- *The family* $\mathbf{\Pi}$ *is* sound *with respect to (X, e, f); that is if there is an accepting computation of the system $\Pi_{|x|}$ on input multiset $e(x)$ then $x \in X$.*
- *The family* $\mathbf{\Pi}$ *is* complete *with respect to (X, e, f); that is, for each input $x \in X$, then every computation of the system $\Pi_{|x|}$ on input multiset $e(x)$ is accepting.*

We define the set of languages decided by a uniform family of membrane systems in polynomial time to be

$$(\mathsf{E}, \mathsf{F})\text{–}\mathsf{PMC}_{\mathcal{R}} = \bigcup_{k \in \mathbb{N}} (\mathsf{E}, \mathsf{F})\text{–}\mathsf{MC}_{\mathcal{R}}(n^k)$$

When the symbols E and F are replaced by complexity class names such as AC^0, L or P it means that the uniformity conditions under consideration are in the function versions of these classes. For example, if we let $\mathsf{E} = \mathsf{F} = \mathsf{AC}^0$ then we mean that the functions $e \in \mathsf{E}$ and $f \in \mathsf{F}$ are computable in uniform FAC^0 and we say we have an AC^0-uniform family.

Let $\mathcal{AM}^0_{+wne}$ denote the class of membrane systems that obey Definition 2, and Definition 1. Thus $(\mathsf{AC}^0, \mathsf{L})$–$\mathsf{PMC}_{\mathcal{AM}^0_{+wne}}$ denotes the class of problems solvable by L-uniform families of active membrane systems without charges in polynomial time with weak non-elementary division rules where the input is encoded using a function in FAC^0.

Remark 1. A membrane system is said to be *confluent* if it is both sound and complete. That is, a membrane system Π is *confluent* if all computations of Π with the same input x (properly encoded) give the same result; either always "accepts" or else always "rejects".

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

3 Polynomial Hierarchy

A well know extension for models of computation is to augment them with an "oracle", that is, the ability to solve certain decision problems in a single timestep. An oracle machine is a machine with access to a special oracle tape that is used to make queries of the form "is $q \in L$" for some language L. By the notation M^{C} we mean the set of problems solved by machines characterising the complexity class M having access to an oracle for a language L in the complexity class C. For instance, P^{NP} is the class of problems solved by deterministic Turing machines working in polynomial time and using an oracle for a problem in NP.

Definition 4 (The Polynomial Hierarchy). *The first level of the hierarchy is* $\Delta_0\mathsf{P} = \Sigma_0\mathsf{P} = \Pi_0\mathsf{P} = \mathsf{P}$. *Then each level of the hierarchy is defined for all* $i \geq 0$*,*

$$\Delta_{i+1}\mathsf{P} = \mathsf{P}^{\Sigma_i\mathsf{P}}$$
$$\Sigma_{i+1}\mathsf{P} = \mathsf{NP}^{\Sigma_i\mathsf{P}}$$
$$\Pi_{i+1}\mathsf{P} = \mathsf{coNP}^{\Sigma_i\mathsf{P}}$$

We define the cumulative polynomial hierarchy to be the class $\mathsf{PH} = \cup_{i\geq 0}\Sigma_i\mathsf{P}$*.*

Note that $\Sigma_1\mathsf{P} = \mathsf{NP}$ and $\Pi_1\mathsf{P} = \mathsf{coNP}$. The hierarchy possesses the following inclusion structure:

$$\Sigma_i\mathsf{P} \cup \Pi_i\mathsf{P} \subseteq \Delta_{i+1}\mathsf{P} \subseteq \Sigma_{i+1}\mathsf{P} \cap \Pi_{i+1}\mathsf{P}, \text{ for all } i \geq 0.$$

Each level of the polynomial hierarchy has its own complete problems.

Problem 1 (Boolean Satisfiability with i Quantifiers (QSAT_d)). Given a Boolean formula φ, and a partitioning of the variables of φ into d sets $X_1, \dots, X_d$. Is there a partial truth assignment for the variables in X_1 such that for all the partial truth assignment for the variables in X_2 such that there is a partial truth assignments for the variables in X_3, and so on up to X_d, such that φ is satisfied by the overall truth assignment?

Lemma 1. QSAT_d *is complete for the class* $\Sigma_d\mathsf{P}$ *[15].*

We have defined QSAT_i so that the odd quantifiers are existential. Without loss of generality we can assume that the expression φ is always in conjunctive normal form with three literals in each clause (3CNF). We refer to this restriction of QSAT_d as $\Sigma_d\mathsf{SAT}$ for short. If the odd numbered sets of variables are universal and φ in disjunctive normal form with 3 variables in each clause (3DNF) we refer to it as $\Pi_d\mathsf{SAT}$.

4 Description of a Uniform Family to Solve $\Sigma_d\mathsf{SAT}$

In this section we provide some details of a uniform family of active membrane systems with a membrane structure $d + 1$ levels deep which decides instances of $\Sigma_d\mathsf{SAT}$. The uniform family implements the following straightforward quantifier elimination algorithm to establish the validity of quantified Boolean formulas. We first describe how the algorithm works on QSAT, then show how it is affected by considering the restriction $\Sigma_d\mathsf{SAT}$. The algorithm works by reducing the problem to the evaluation of quantifier-free and variable-free expressions. This method is based on the following simple observations:

$$\begin{aligned} \forall x \psi(x) &\iff \psi(0) \wedge \psi(1) \\ \exists x \psi(x) &\iff \psi(0) \vee \psi(1). \end{aligned}$$

By applying these equivalences recursively to an instance of $\Sigma_d\mathsf{SAT}$, the quantifiers can be eliminated one by one. We then evaluate the final fully expanded expression to obtain the result. This evaluation can be computed in polynomial time with respect to the size of the expression; note however, that the expression to evaluate is exponentially larger than the input, since eliminating a quantifier doubles its size.

This quantifier elimination algorithm is needlessly inefficient when executed sequentially: since QSAT is in PSPACE, this problem can be solved in exponentially less space. However, the algorithm can be made to run in polynomial time if an exponential number of processors are available. The two sub-formulas resulting from the elimination of a quantifier can be evaluated *independently*, and their truth values conjuncted or disjuncted (according to the specific quantifier) only in the last step. This is equivalent to evaluating the formula under every possible assignment to the variables, then feeding the results into an exponentially-sized

Boolean circuit C which forms a complete binary tree (the same form as the recursion tree of the quantifier elimination algorithm, or equivalently, as the parse tree of the resulting Boolean expression) where the nodes of depth i are $\wedge$-gates (resp., $\vee$-gates) if variable x_{i+1} is universally (resp., existentially) quantified. Notice that the depth of this circuit is linear with respect to the number of variables.

When the number of alternations of quantifiers is bounded by a constant d (as in the problems $\Sigma_d\mathsf{SAT}$ and $\Pi_d\mathsf{SAT}$), and if unbounded fan-in gates are available, the circuit used to combine the results of the evaluation of the formula can be reduced to constant depth d. Indeed, a sequence of k consecutive quantifiers can be eliminated simultaneously, as long as they are all universal or all existential, and the values of the 2^k resulting sub-formulas fed into a single $\wedge$- or $\vee$-gate, thus increasing the depth of the circuit just by one. In symbols:

$$\forall x_1 \cdots \forall x_k \varphi(x_1, \ldots, x_k) \iff \bigwedge_{(x_1,\ldots,x_k)\in\{0,1\}^k} \varphi(x_1, \ldots, x_k)$$
$$\exists x_1 \cdots \exists x_k \varphi(x_1, \ldots, x_k) \iff \bigvee_{(x_1,\ldots,x_k)\in\{0,1\}^k} \varphi(x_1, \ldots, x_k).$$

4.1 Encoding of $\Sigma_d\mathsf{SAT}$ instances

We specify that instances of $\Sigma_d\mathsf{SAT}$ are encoded as follows.

We encode which variables are bound by which quantifiers in a binary matrix Q with m rows and m columns. Each column represents one of the m variables of the formula. There are a maximum of m rows since at most $d \leq m$ quantifiers are possible for each instance. The elements of Q are defined as follows:

$$q_{i,j} = \begin{cases} 1 & \text{variable } x_j \text{ is bound by the } i^{\text{th}} \text{ quantifier} \\ 0 & \text{otherwise} \end{cases}$$

To encode the Boolean formula φ we use P a $2m \times 2m \times 2m$ three dimensional binary matrix. Each element of the matrix represents the three variables in a clause in the problem instance. If the element $q_{i,j,k} = 1$ then the variables $x_{i \bmod m}$, $x_{j \bmod m}$ $x_{k \bmod m}$ exist in the clause. If $i < m$ then the variable x_1 is unnegated in the clause while if $i > m$ then the variable x_i appears negated.

The total length of the binary string (we flatten the matrices to strings) to encode an instance of $\Sigma_d\mathsf{SAT}$ with m variables is thus $m^2 + 2m^3$ bits.

4.2 Evaluating quantified Boolean formulas

We now describe a logspace uniform family of active membrane systems without charges to recognise problem $\Sigma_d\mathsf{SAT}$ and where d is odd. (The arguments for even d and for the problem $\Pi_d\mathsf{SAT}$ are similar.) The family encoding function f takes as input the number $1^{m^2+2m^3}$ which is the length of the input instance encoded in unary, from this it calculates the value m which is used to construct the family member Π_n.

We present a high level sketch of the membrane structure and rules of Π_n to convince the reader of the systems existence.

The membrane structure μ of Π_n is represented (in bracket language) as follows:

$$\overbrace{\Big[\Big[\cdots\Big[\Big[}^{d\text{ membranes}}\ \Big[\underbrace{[\,]_{c_{\langle 1,1,1\rangle}}[\,]_{c_{\langle 1,1,2\rangle}}\cdots[\,]_{c_{\langle 2m,2m,2m\rangle}}}_{2m^3\text{ membranes}}\Big]_{d+1}\ \Big]_d\Big]_{d-1}\cdots\Big]_2\Big]_1$$

The input membrane is $d+1$ and contains the objects produced by the e function from Definition 3. This function takes a potential instances of $\Sigma_d\mathsf{SAT}$ as input, instances are encoded as binary strings using the scheme described in Section 4.1. For each element $p_{i,j,k}=0$ of the matrix P an object $\overline{c}_{\langle i,j,k\rangle}$ is created, these represent the clauses *not* used in the instance. For each element $q_{i,j}=1$ of the matrix Q the object $x_{i,j,0}$ such that $1 \le i \le m$ and variable x_i is bound by the j-th quantifier in the input formula. The third subscript of $x_{i,j,0}$ is a time counter, which is incremented during each computation steps by evolution rules such as $[x_{i,j,t} \to x_{i,j,t+1}]_{d+1}$, unless a different behaviour is explicitly described below for some values of t. It is easy to imagine how a uniform constant depth circuit can map the encoding described in Section 4.1 to these objects, so we claim the object encoding function e is in FAC^0.

A timer-object $z_{i,0}$ is contained in membrane i for $1 \le i \le d+1$; it is also incremented via $[z_{i,t} \to z_{i,t+1}]_i$ during each step, unless explicitly stated below. All membranes c_j contain an analogous object $z_{0,0}$.

The computation of Π_n on a given input is divided into four phases.

Phase 1. Dissolution of membranes representing unused clauses.

Membrane $c_{\langle i,j,k\rangle}$ represents the same clause as the element $p_{i,j,k}$ in Section 4.1. These membranes are dissolved during the first two computation steps if that clause does not occur in the input formula (i.e., if object $\overline{c}_{\langle i,j,k\rangle}$ occurs in the input multiset), according to the following rules:

$$\overline{c}_{\langle i,j,k\rangle}\,[\,]_{c_{\langle i,j,k\rangle}} \to [\overline{c}_{\langle i,j,k\rangle}]_{c_{\langle i,j,k\rangle}} \qquad [\overline{c}_{\langle i,j,k\rangle}]_{c_{\langle i,j,k\rangle}} \to \lambda$$

The total duration of Phase 1 is exactly two computation steps.

Phase 2. Quantifier elimination

In the second phase we use non-elementary membrane division in order to carry out the process of quantifier elimination, as described in the beginning of this section.

Let x_i be the variable bound by the first quantifier having the smallest subscript. The corresponding object $x_{i,1,2}$ (here the third subscript is 2 because the first phase took two steps) is first moved to membrane 2 (the one immediately below the corresponding quantifier-membrane) by using a series of communication rules:

$$[x_{i,1,2}]_{d+1} \to [\,]_{d+1}\; x_{i,1,3} \qquad [x_{i,1,3}]_{d} \to [\,]_{d}\; x_{i,1,4} \qquad \cdots$$
$$[x_{i,1,d-1}]_{4} \to [\,]_{4}\; x_{i,1,d} \qquad [x_{i,1,d}]_{3} \to [\,]_{3}\; x_{i,1,d+1}$$

Then, object $x_{i,1,d+1}$ divides membrane 2, *duplicating all of its substructure*, and becoming a “true” object on one side and a “false” object on the other:

$$[x_{i,1,d+1}]_2 \to [t_{i,\epsilon,d+2}]_2\; [f_{i,\epsilon,d+2}]_2$$

The second subscript is erased (i.e., replaced by ϵ) in the process, since it is not needed anymore. The object $t_{i,\epsilon,d+2}$ is now brought back to membrane $d+1$ using another series of communication rules:

$$t_{i,\epsilon,d+2}\; [\,]_3 \to [t_{i,\epsilon,d+3}]_3 \qquad t_{i,\epsilon,d+3}\; [\,]_3 \to [t_{i,\epsilon,d+4}]_3 \qquad \cdots$$
$$t_{i,\epsilon,2d}\; [\,]_d \to [t_{i,\epsilon,2d+1}]_d \qquad t_{i,\epsilon,2d+1}\; [\,]_{d+1} \to [t_{i,\epsilon,2d+2}]_{d+1}$$

and analogously for $f_{i,\epsilon,d+2}$. Notice that now we have *two* instances of membrane $d+1$: in one of them, the variable x_i is set to true, and in the other it is set to false. The timer subscript of $t_{i,\epsilon,d+2}$ and $f_{i,\epsilon,d+2}$ continues to be incremented inside membrane $d+1$.

In the subsequent steps, the objects representing the other variables bound by the first quantifier move to membrane 2 to divide and generate an assignment for their variable then move back to membrane $d+1$. This same process is then performed for all variables bound by the second quantifier, then third and so on until the d^{th}quantifier. The timers can be synchronized correctly by always assuming the longest possible path (from membrane $d+1$ to 2) which is $2d+1$ steps. The time required by Phase 2 is then $m(2d+1)$ steps. At the end of this phase each of the 2^m copies of membrane $d+1$ contains a different assignment (either a $t_{i,\epsilon,m(2d+1)+2}$ or $f_{i,\epsilon,m(2d+1)+2}$ object) to the variables $x_1, \ldots, x_m$.

Phase 3. Evaluation of the matrix of the formula.

The objects representing truth assignments of the variables now must be copied so that there are enough for each clause-membrane to take in. That is, each membrane representing a clause containing the literal x_i can bring in a copy of the corresponding “true” object, and each one containing the literal $\bar{x}_i$ can bring in a copy of the corresponding “false” object. Each of the copies is subscripted by the name of one of the clauses which they satisfy, i.e.,

$$[\; t_{i,\epsilon,m(2d+1)+2} \to \{t'_{i,\epsilon,\langle j,k,l\rangle} \mid i = j \vee i = k \vee i = l\} \;]_{d+1}$$
$$[\; f_{i,\epsilon,m(2d+1)+2} \to \{f'_{i,\epsilon,\langle j,k,l\rangle} \mid i+m = j \vee i+m = k \vee i+m = l\} \;]_{d+1}$$

Notice that these objects do not need a timer subscript.

To evaluate a clause occurring in the input formula, membrane $c_{\langle i,j,k\rangle}$ tries to bring in one of the objects corresponding to a variable assignment that will make the clause true. (Recall that in the first phase we removed all clauses not appearing in the input instance.) For example, for the membrane $c_{\langle 1,2,6\rangle}$ which represents the clause $x_1 \vee x_2 \vee \bar{x}_3$ uses the following rules:

$$\begin{aligned} t'_{1,\epsilon,\langle i,j,k\rangle}\,[\,]_{c_{\langle 1,2,6\rangle}} &\to [t'_{d+1}]_{c_{\langle 1,2,6\rangle}} \text{ where } 1 = i \vee 1 = j \vee 1 = k \\ t'_{2,\epsilon,\langle i,j,k\rangle}\,[\,]_{c_{\langle 1,2,6\rangle}} &\to [t'_{d+1}]_{c_{\langle 1,2,6\rangle}} \text{ where } 2 = i \vee 2 = j \vee 2 = k \\ f'_{3,\epsilon,\langle i,j,k\rangle}\,[\,]_{c_{\langle 1,2,6\rangle}} &\to [t'_{d+1}]_{c_{\langle 1,2,6\rangle}} \text{ where } 6 = i \vee 6 = j \vee 6 = k \end{aligned}$$

At most three objects are sent to in each clause membrane $c_{\langle i,j,k\rangle}$ in successive steps. One of the objects t'_{d+1} (whose only subscript indicates that it is a "true" object of level $d+1$, this prevents mixing up truth values on different levels of the membrane structure) after at most three steps dissolves membrane $c_{\langle i,j,k\rangle}$ via

$$[t'_{d+1}]_{c_{\langle i,j,k\rangle}} \to t'_{d+1}$$

If t'_{d+1} has not dissolved $c_{\langle i,j,k\rangle}$ at time $m(2d+1)+7$, then we infer that the clause is not satisfied. The counter object $z_{0,m(2d+1)+7}$ (whose second subscript has been incremented each step) in each remaining clause membrane evolves into a false value and dissolves the membrane:

$$[z_{0,m(2d+1)+7}]_{c_{\langle i,j,k\rangle}} \to f'_{d+1}$$

After at most six computation steps, all the objects denoting the results of the evaluations have been sent to $d+1$.

Membrane $d+1$ now computes the conjunction of the value-objects located inside it. If a "false" object f'_{d+1} appears, then the whole conjunction has a false result which is denoted by f'_d:

$$[f'_{d+1}] \to f'_d$$

If no instance of f'_{d+1} appears inside $d+1$ at time $m(2d+1)+9$, then all clauses evaluate to true. The object $z_{d+1,m(2d+1)+9}$ (obtained by repeatedly increasing the second subscript of $z_{d+1,0}$ as described above for $z_{0,0}$) is then used to produce a true result:

$$[z_{d+1,m(2d+1)+9}] \to t'_d$$

Now each instance of membrane d contains either t'_d or f'_d, each of these objects represents the evaluation of Boolean formula on some assignment. The whole phase requires at most eight steps.

Phase 4. Computing the value of the whole formula

The 2^{m-1} copies of membrane d (corresponding to the innermost quantifier of the input formula) must now combine the results coming from the (now dissolved) children membranes labelled by $d+1$. Since d is odd by hypothesis, the last quantifier is $\exists$, hence the results must be combined by disjunction. If a true object t'_d exists, then the result of the evaluation is, in turn, true:

$$[t'_d]_d \to t'_{d-1}$$

otherwise, we can dissolve d via the object $z_{d,m(2d+1)+11}$ (when its counter reaches this value), which is transformed into a false value:

$$[z_{d,m(2d+1)+11}]_d \to f'_{d-1}$$

The evaluation then proceeds to the upper (i.e., outermost) levels of the membrane structure in a completely analogous way, alternating universal quantification (corresponding to conjunction, which is performed as described at the end of Phase 3) and existential quantification. Clearly, the counters of the $z_{i,t}$ objects must be adjusted appropriately: this is easy to accomplish, since the evaluation of each quantifier requires at most two computation steps.

The only difference occurs on the last level: instead of sending out object t'_0 or f'_0 from the outermost membrane, we use the objects `yes` and `no` in order to signal the result of the whole computation.

Phase 4 is completed in $2d$ steps.

This Section describes the proof for the following theorem.

Theorem 1. $\Sigma_d\mathsf{SAT} \in (\mathsf{AC}^0, \mathsf{L})\text{–}\mathsf{PMC}_{\mathcal{AM}^0_{+wne}}$ *where the depth of the membrane structure is limited to* $d+1$.

Note that we do not give the membrane system family in enough detail to show that it holds for AC^0-uniformity however the system described is easily L uniform.

Corollary 1. $\mathsf{QSAT} \in (\mathsf{AC}^0, \mathsf{L})\text{–}\mathsf{PMC}_{\mathcal{AM}^0_{+wne}}$ *if the depth of the membrane structure is polynomial of* m*, the number of variables.*

5 Conclusions and Future Directions

We proved that in the setting of active membrane systems without charges and using non-elementary division rules, a membrane structure of depth $d+1$ is sufficient to decide (in polynomial time) the validity of quantified Boolean formulas with d alternations of quantifiers. An interpretation of this result is that each level of nesting of membranes provides access to an oracle. Since there is no known way to perform the same task using substantially shallower membrane structures, this seems to suggest that increasing the depth of the membrane structure actually increases the computing power of the systems.

Whether this apparent phenomenon corresponds to reality remains an open problem. Future work on this topic may involve simulating arbitrary $(d+1)$-depth families of membrane systems by devices characterising the d^{th} level of the polynomial hierarchy (such as suitable alternating Turing machines). Also, identifying the relationship between depth and membrane division on the one hand, and alternation on the other.

References

1. A. Alhazov, M.J. Pérez-Jiménez. Uniform solution to QSAT using polarizationless active membranes. In J. Durand-Lose, M. Margenstern, editors, *Machines, Computations and Universality (MCU)*, LNCS 4664, pages 122–133, Orléans, France, September 2007.
2. D.A. Mix Barrington, N. Immerman, H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
3. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. A logarithmic bound for solving subset sum with P systems. In *8th International Workshop on Membrane Computing, WMC 2007*, LNCS 4860, pages 257–270. Springer, 2007.
4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Néñez, F.J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics*, 83(7):593–611, 2006.
5. G. Mauri, M. Pérez-Jiménez, C. Zandron. On a Păun's conjecture in membrane systems. In José Mira and José R. Álvarez, editors, *Bio-inspired Modeling of Cognitive Tasks*, LNCS 4527, pages 180–192. Springer, Berlin, 2007.
6. N. Murphy, D. Woods. Active membrane systems without charges and using only symmetric elementary division characterise P. In G. Eleftherakis et al., editors, *Membrane Computing, 8th International Workshop, WMC 2007 Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*, LNCS 4860, pages 367–384. Springer, 2007.
7. N. Murphy, D. Woods. A characterisation of NL using membrane systems without charges and dissolution. In *UC*, pages 164–176. Springer, 2008. To appear.
8. N. Murphy, D. Woods. Uniformity conditions in natural computing. In *Proceedings of DNA16*, 2010. To appear.
9. Gh. Păun. P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
10. Gh. Păun. *Membrane Computing*. Springer, Berlin, 2002.
11. M.J. Pé rez-Jimé nez, A. Romero-Jiménez, F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–285, 2003.
12. M.J. Pérez-Jiménez, A. Riscos-Núñez, A. RomeroJiménez, D. Woods. *Handbook of Membrane systems*, chapter 12: Complexity Membrane Division, Membrane Creation. Oxford University Press, 2009.
13. A.E. Porreca, A. Leporati, C. Zandron. On a powerful class of non-universal P systems with active membranes. Submitted, 2010.
14. P. Sosík, A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73(1):137–152, 2007.
15. L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
16. D. Woods, N. Murphy, M.J. Pérez-Jiménez, A. Riscos-Núñez. Membrane dissolution and division in P. In *Unconventional Computation*, volume 5715, pages 262–276, 2009.

On Complexity Classes of Spiking Neural P Systems

Alfonso Rodríguez-Patón[1], Petr Sosík[1,2], Luděk Cienciala[2]

[1] Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain

[2] Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava, 74601 Opava, Czech Republic

Summary. A sequence of papers have been recently published, pointing out various intractable problems which may be solved in certain fashions within the framework of spiking neural (SN) P systems. On the other hand, there are also results demonstrating limitations of SN P systems. In this paper we define recognizer SN P systems providing a general platform for this type of results. We intend to give a more systematic characterization of computational power of variants of SN P systems, and establish their relation to standard complexity classes.

1 Introduction

The spiking neural P systems, incorporating in membrane computing ideas from spiking neurons, see, e.g., [18], were introduced in [12]. In short, an SN P system consists of a set of neurons placed in the nodes of a graph, representing synapses. The neurons send signals (spikes) along synapses (edges of the graph). This is done by means of firing rules, which are of the form $E/a^c \to a; d$, where E is a regular expression, c is the number of spikes consumed by the rule, and d is the delay from firing the rule and emitting the spike. The rule can be used only if the number of spikes collected by the neuron is "covered" by expression E, in the sense that the current number of spikes in the neuron, n, is such that $a^n \in L(E)$, where $L(E)$ is the language described by expression E. In the interval between firing a rule and emitting the spike, the neuron is closed/blocked, it does not receive other spikes and cannot fire again. There also are rules for forgetting spikes, of the form $a^s \to \lambda$ (s spikes are just removed from the neuron). Starting from an initial distribution of spikes in the neurons and using the rules in a synchronized manner (a global clock is assumed), the system evolves. A sequence of transitions among configurations of an SN P system, starting in the initial configuration, is called a computation. One of the neurons is designated as the output neuron and its spikes can also exit the system. The sequence of steps when the output neuron sends spikes to the environment is called the spike train of the computation.

In section 3.2 we propose a definition of (uniform) families of recognizer SN P systems which should be still more elaborated in the future work. Then we use this apparatus to study the computational power of deterministic (or, more generally, *confluent*) SN P systems under the following conditions:

general regular expressions / single-star normal form,
polynomial / exponential number of activated neurons,
polynomial uniformity / non-uniformity by Turing machines,
cyclic / acyclic architecture.

Main theoretical tools we use to study these topics are logic circuits and RAM computers which can simulate SN P systems in a convenient way.

2 Prerequisites

We assume the reader to be familiar with basic language and automata theory, as well as with basic membrane computing, e.g., from [22] and [26], respectively (we also refer to [21] for an up-to-date information about membrane computing), so that we introduce here only some notation used later in proofs.

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

For definitions of boolean circuits we refer the reader to [2]. Let $\mathbf{UCKT}(C(n), D(n))$, where $C(n) \geq n$, denote the class of problems solvable by logspace-uniform circuit families with unbounded fan-in where element corresponding to n has size $\mathcal{O}(C(n))$ and depth $\mathcal{O}(D(n))$.

2.1 Regular expressions and normal forms

We recall the definition of regular expression mostly in order to fix the notation.

Definition 1. *For a finite alphabet V : (i) λ and each $a \in V$ are regular expressions, (ii) if E_1, E_2 are regular expressions over V, then also $(E_1) \cup (E_2), (E_1)(E_2)$, and $(E_1)^*$ are regular expressions over V, and (iii) nothing else is a regular expression over V.*

The catenation operator $\cdot$ and non-necessary parentheses may be omitted when writing a regular expression. With each expression E we associate its language $L(E)$ defined in a usual way. We call two expressions E_1 and E_2 *equivalent* if $L(E_1) = L(E_2)$.

Definition 2 ([1]). *We say that a regular expression $E = E_1 \cup \ldots \cup E_n$ (where each E_i contains only $\cdot$ and $*$ operators) is in* single-star normal form *(SSNF) if* $\forall i \in \{1, \ldots, n\}$, *$E_i$ has at most one occurrence of $*$.*

Lemma 1 ([1]). *Every regular expression over one-letter alphabet can be transformed into an equivalent single-star normal form.*

This transformation, however, might require an exponential time and the size of the resulting expression can be exponential with respect to the size of the original expression.

2.2 Random Access Machines

RAM is a computing device with an infinite random access array of data registers $M_1, M_2, \ldots$, each of which can store an arbitrary integer, and with a set of labelled instructions $P = \{P_1, P_2, \ldots\}$ each from the instruction set described below.

$constant \longrightarrow M_{res}$	
$M_{op1} \longrightarrow M_{res}$	
$*M_{op1} \longrightarrow M_{res}$	The contents of a register whose address is stored in register M_{op} - indirect memory READ
$M_{op1} \longrightarrow *M_{res}$	The contents of a register M_{op} is written into the register whose address is in register M_{res} - indirect memory WRITE
$M_{op1} + M_{op2} \longrightarrow M_{res}$	
$M_{op1} - M_{op2} \longrightarrow M_{res}$	
GOTO label	
GOTO label if $M_{op1} @ M_{op2}$ $@ \in \{=, <\}$	
HALT	

Non-deterministic RAM is defined in the same manner as its deterministic version except that more than one instruction can have the same label and their choice is non-deterministic.

A parallel RAM (PRAM) has a sequence of RAM's $R_1, R_2, \ldots$ operating synchronously in parallel. They all have their own local registers and they can read and write to common registers too. This can be done by indirect memory READ or WRITE. In CRCW PRAM (concurrent-read, concurrent-write) if more than one processor attempts to write into the same location in common memory at the same time, the lowest numbered processor succeeds. All processors run the same program.

In addition to the programs, another part of specification of a particular CRCW PRAM is a function $P(n)$ from positive integers to positive integers called the *processor bound*. An input of size n consists of n binary words, each of length at most n. A CRCW PRAM is given an input of size n by placing n words in the first n locations of common memory, and the first $P(n)$ processors $R_1, \ldots, R_{P(n)}$

are started. Each instruction takes one time unit. The computation halts when $R_1, \ldots, R_{P(n)}$ have all halted. The machine operates in time $T(n)$ if it halts within $T(n)$ steps on every input of size n. When the computation halts, the output can be found in initial contiguous block of common memory of length at most n. The model is essentially identical to the SIMDAG of Goldschlager [8] and similar to the P-RAM of Fortune and Wyllie [7]. Note that [8] and [7] characterized the power of these models when the number of processors grows exponentially in n.

Denote by $\mathbf{CRCW}(P(n), T(n))$ the class of problems solvable on CRCW PRAM in $\mathcal{O}(T(n))$ time with $\mathcal{O}(P(n))$ processors. It is known that $\mathbf{CRCW}(poly(n), poly(n)) = \mathbf{P}$ while $\mathbf{CRCW}(exp(n), poly(n)) = \mathbf{PSPACE}$.

Lemma 2 ([28]). $\mathbf{CRCW}(P(n), T(n)) \subseteq \mathbf{UCKT}(poly(P(n), T(n), n), T(n))$

The above result was presented first in [28] where also its detailed proof together with constructions of logical circuits implementing PRAM can be found. The formulation in Lemma 2 is inspired by [14].

3 Spiking Neural P Systems

A *spiking neural membrane system* (abbreviated as SN P system), of degree $m \geq 1$, is a construct of the form

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

where:

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

   $$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

   where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \to a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
      (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicates the *input neuron* (resp., *output neuron*).

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a; d$ can be applied. The application of this rule means consuming (removing) c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces a spike after d time units (as usual in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost). In the step $t+d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t+d+1$).

The rules of type (2) are *forgetting* rules; they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

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

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used. Since two firing rules, $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and in that case, only one of them is chosen non-deterministically. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The system is called *deterministic* if for every neuron that occurs in the system, any two rules $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$ in the neuron have $L(E_1) \cap L(E_2) = \emptyset$.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron. During a computation, the "state" of the system is described by both by the number of spikes present in each neuron, and by the open/closed condition of each neuron: if a neuron is closed, then we have to specify when it will become open again.

Using the rules as described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation.* A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation (halting or not) we associate a *spike train*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

3.1 Descriptional complexity of SN P systems

The size of description of a SN P system is an important measure when one wants to study families of SN P systems. The first detailed specification of descriptional size of a SN P system Π is given in [16]. It is based on the number of bits necessary to fully describe the system Π. Let m be the number of neurons, N be the maximum natural number that appears in the definition of Π, R the maximum number of rules which occur in its neurons, and S the maximum size required by the regular expressions that occur in Π (this will be discussed later). Then the total size of description of Π is polynomial with respect to m, R, S and $\log N$.

Some authors introducing space complexity measures for P systems as [25] suggested that an n-tuple of identical objects should occupy a space n. In SN P systems, however, the spikes are not physical objects but an electric potential. Furthermore, if we decided for unary representation of spikes, then it would be also fair to assume unary representation of regular expressions in neurons which, however, could restrict their power significantly.

Finally, observe that classical models of P systems and their uniform families (see the next section for definition of families) actually use binary representation of data. A member of the family processing inputs of size n can use $poly(n)$ different objects and code its input by their (non)presence in the input membrane.

Therefore, to represent n spikes, just the number $\log n$ of bits is taken as the size of its description, both in neurons and regular expressions. This succinct representation is used also in [16] and other papers. Note, however, that many of the results presented here remain valid even if the unary representation of spikes were adopted.

3.2 Families of SN P systems solving decision problems

Standard SN P systems were shown to be universal already in the introductory paper [12]. However, as demonstrated in [20], no standard spiking neural P system with a constant number of neurons can simulate Turing machines with less than exponential time and space overheads. Therefore, for application of SN P systems to solve intractable problems, many authors have (implicitly) used families of SN P systems such that each member of a family solves only a finite set of instances of a given size. In this section we propose a formal specification for families of SN P systems. A first step towards such a specification was taken already in [15] for the case of *non-deterministic* (and non-confluent) SN P systems. Most of definitions in this section is motivated by [23].

Let us call *decision problem* a pair $X = (I_X, \theta_X)$ where I_X is a language over a finite alphabet (whose elements are called instances) and θ_X is a total boolean function over I_X.

It was suggested by some authors that a SN P systems solving an instance $w \in I_X$ would halt if and only if $\theta_X(w) = 1$. However, this is incompatible with definitions of basic complexity classes (which do not allow non-halting computations) and also with standard construction of families of recognizer P systems [23] which is extensively used. Hence we suggest the convention used implicitly by many authors ([9, 16] and others) when describing “neural” solutions to SUBSET SUM, 3-SAT and similar problems: the system always halts and the result is determined by the fact whether the output neuron emits a spike during the computation. Other convention are possible, such as the existence of two output neurons signalling *true* or *false*.

Definition 3. *A* recognizer SN P system *is a SN P system which has all computations halting, and whose output neuron spikes no more than once during each computation. Its computation is called* accepting *if the output neuron spikes exactly once, otherwise it is* rejecting*.*

This definition is also compatible with the variant when the system is asked to spike *at least once* in the case of accepting computation. Any such SN P system can be added another neuron connected to the original output, with two initial spikes and the rules $a \to \lambda$ and $a^3 \to a; 0$ which emits only the first spike of those it receives.

We distinguish recognizer SN P systems with input or without input. In the first case the input (i.e., an instance $w \in I_X$) is sent in the form of binary spike train $bin(w)$ to the input neuron, where $bin(w)$ is an arbitrary *binary* encoding of w. There are reasons for choosing binary encoding: it is known that standard SN P systems can simulate logic gates with unbounded fan-in in a unit time [9]. Hence, their computational potential is at least as high as that of logic circuits. The unary input/output convention, however, would decrease their computational power *exponentially* in many cases, on one hand. On the other hand, to transform the input value into the number of spikes in a neuron, the extended rules or maximal parallelism would be necessary [17].

In the case of SN P systems without input, instances of a problem are encoded within the structure of SN P system.

Definition 4. *A family* $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ *(respectively,* $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$*) of recognizer SN P without input (resp., with input) is* polynomially uniform by Turing machines *if there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *(resp.,* $\Pi(n)$*) from the instance* $w \in I_X$ *(resp., from* $n \in \mathbb{N}$*).*

In the sequel we will for short denote such a family just as *uniform.* Necessary conditions must be met by families of recognizer SN P systems to solve algorithmically a given decision problem. Conditions of *soundness* and *completeness* of $\mathbf{\Pi}$ with respect to X are defined in [23]. Conjunction of these two conditions for SN P systems without input (resp., with input) ensures that for every $w \in I_X$, if $\theta_X(w) = 1$, then every computation of the SN P systems solving w is accepting, and if $\theta_X(w) = 0$, then every such computation is rejecting. Note that this SN P system can be generally nondeterministic, i.e, it may have different possible computations, but with the same result. Such a P system is also called *confluent.*

Definition 5. *Let* $f : \mathbb{N} \to \mathbb{N}$ *be a constructible function. A decision problem* X *is solvable in time bounded by* f *by a family* $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ *of recognizer SN P systems of type* $\mathcal{R}$ *without input, denoted by* $X \in \mathbf{SN}^*_{\mathcal{R}}(f)$*, if the following holds:*

- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines.*
- *The family* $\mathbf{\Pi}$ *is* f *time-bounded; that is, for each instance* $w \in I_X$*, every computation of* $\Pi(w)$ *performs at most* $f(|w|)$ *steps.*

- *The family* $\mathbf{\Pi}$ *is sound and complete with respect to* X.

The family $\mathbf{\Pi}$ is said to provide a *semi-uniform solution* to the problem X. In this case, for each instance of X we have a special P system. Specifically, we denote by

$$\mathbf{PSN}^*_{\mathcal{R}} = \bigcup_{f \text{ polynomial}} \mathbf{SN}^*_{\mathcal{R}}(f)$$

the class of problems to which uniform families of SN P systems of type $\mathcal{R}$ without input provide semi-uniform solution in polynomial time. Analogously we define families which provide uniform solutions solutions to decision problems.

Definition 6. *Let* $f : \mathbb{N} \to \mathbb{N}$ *be a constructible function. A decision problem* X *is solvable in time bounded by* f *by a family* $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ *of recognizer SN P systems of type* $\mathcal{R}$ *with input, denoted by* $X \in \mathbf{SN}_{\mathcal{R}}(f)$, *if the following holds:*

- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines.*
- *The family* $\mathbf{\Pi}$ *is* f *time-bounded; that is, for each instance* $w \in I_X$, *every computation of the member of* $\mathbf{\Pi}$ *solving* w *performs at most* $f(|w|)$ *steps.*
- *The family* $\mathbf{\Pi}$ *is sound and complete with respect to* X.

The family $\mathbf{\Pi}$ is said to provide a *uniform solution* to the problem X. Again, we denote by

$$\mathbf{PSN}_{\mathcal{R}} = \bigcup_{f \text{ polynomial}} \mathbf{SN}_{\mathcal{R}}(f)$$

the class of problems to which uniform families of SN P systems of type $\mathcal{R}$ with input provide uniform solution in polynomial time. Obviously, for any constructible function f and a class of SN P systems $\mathcal{R}$, $\mathbf{SN}_{\mathcal{R}}(f) \subseteq \mathbf{SN}^*_{\mathcal{R}}(f)$, and $\mathbf{PSN}_{\mathcal{R}} \subseteq \mathbf{PSN}^*_{\mathcal{R}}$.

We use the following notation to describe a specific type $\mathcal{R}$ of SN P systems: $-reg$ for systems with regular expressions of the form a^n, $n \geq 1$, $-del$ for systems without delays, and $ssnf$ for systems with regular expressions in the single-star normal form.

4 Simulation of SN P systems with RAM

Theorem 1. *For each confluent (respectively, non-confluent) SN P system* Π *with all regular expressions in single-star normal form (SSNF) and with description of size* s, *there is a deterministic (resp., non-deterministic) RAM constructed in polynomial time with unit costs of operations which simulates* t *steps of* Π *in time* $\mathcal{O}(t(s+t))$.

Proof. All information about the topology of the SN P system and rules in neurons can be contained within the simulating RAM program. The configuration of each neuron of a SN P systems will be represented in RAM by three registers counting

(i) the number of spikes in the neuron, (ii) current delay period of the neuron (i.e., the remaining number of steps during which the neuron will be closed), and (iii) the number of spikes prepared to be emitted (0 or 1, but in case of *extended* SN P systems it can be more). Simulation of single step of a SN P system with RAM consists of the following phases:

1. For each neuron:
   - If the delay period is 0, check whether the number of spikes in neuron is covered by a regular expression in some of its rules. If more than one rule matches, choose one randomly. If a rule $E/a^c \to a; d$ or $a^c \to \lambda$ can be applied, then
     a) decrease the number of spikes in neuron by c,
     b) set the delay period to d in the case of firing rule,
     c) set the number of spikes to be emitted to 1 for firing rule or to 0 for erasing rule.
   - Else decrement the delay period by 1.
2. For each neuron with delay period 0 and the number of spikes to be emitted ¿0: reset the number of spikes to be emitted and increase by 1 the number of spikes in all neurons connected to the output of the processed neuron.
3. If all neurons have delay period 0, none rule was applied and none spike was emitted, halt the computation.

It is easy to verify that almost each elementary operation described above for a single neuron or a synapse can be implemented on RAM in $\mathcal{O}(1)$ time, hence the total number of necessary RAM operations for all neurons is $\mathcal{O}(s)$.

The only exception is the evaluation whether the number of spikes in a neuron is covered by regular expressions. Consider a regular expression $E = E_1 \cup \ldots \cup E_n$ in SSNF. Each E_i, $1 \leq i \leq n$, can be easily rewritten to the form $a^q(a^r)^*$, for $q, r \geq 0$. The evaluation whether E_i covers a number k of spikes present in the neuron consist of checking the divisibility of $k-q$ by r. This operation is performed in time proportional to the number of bits of k. Total number of bits to describe all spikes in all neurons after t steps of computation is $\mathcal{O}(s+\log t)$ for standard SN P systems, or $\mathcal{O}(s+t)$ in the case of maximal parallelism or exhaustive rules. Hence, the total number of RAM operations necessary to simulate t steps is $\mathcal{O}(t(s+t))$.

The situation is more complicated when we deal with general regular expressions in neurons. Let us prove the following lemma first.

Lemma 3. *Matching of a regular expression E of size s in succinct form over a singleton alphabet with a string a^k can be done on a RAM or a Turing machine in non-deterministic polynomial time with respect to $s \log k$.*

Proof. Assume that we have the syntactic tree of the expression E at our disposal (its parsing can be done in deterministic polynomial time). We treat the subexpressions of the form a^n as constants and assign them a leaf node of the tree with the value n. The matching algorithm works as follows:

- Produce non-deterministically a random element of $L(E)$ in succinct form by a depth-first search traversal of its syntactic tree. Start with the value 0 and evaluate recursively each node depending on its type as follows:
  - leaf node containing a constant: return the value of the node;
  - catenation: evaluate both subtrees of this node and add the results;
  - union: choose non-deterministically one of the subtrees of this node and evaluate it;
  - star: draw a random number of iterations x within the range $\langle 0, k\rangle$, evaluate the subtree starting in this node and multiply the result by x.
- Compare the drawn element of $L(E)$ with a^k whether they are equal.

Whenever during the evaluation the computed value exceeds k, the algorithm halts immediately and reports that a^k does not match $L(E)$. This guarantees that the number of bits processed in each operation is always $O(\log k)$.

Each of the elementary operations described above can be performed in constant time on RAM with unit instruction price, except the multiplication which requires $O(\log k)$ time. Total number of tree-traversal steps is $\mathcal{O}(s)$. If we implement the algorithm on Turing machine, the resulting time increases only polynomially.

Theorem 2. *Any confluent or non-confluent SN P system Π of size s with general regular expressions which performs t steps can be simulated on a RAM in polynomial space with respect to t and s.*

Proof. Consider the confluent case first. The construction of RAM used in the proof of Theorem 1 can remain in this proof unchanged except the problem of matching a number of spikes in a neuron with a general regular expression E over one letter alphabet. Lemma 3 states that this problem can be solved in a non-deterministic polynomial time on a RAM. Therefore, it might seem that the whole simulation of a SN P systems might belong to the class **NP**. However, it is not the case (unless **NP**=**co-NP**). The problem is that we cannot iterate the non-deterministic solutions guessing whether certain neuron spikes, since the overall result of computation might depend of the fact whether some neurons *do* spike and others *do not* spike.

Therefore, since **NP** $\subseteq$ **PSPACE**, we can only guarantee that the whole simulation can be done in deterministic polynomial space. Indeed, if one replaces the random selection in the proof of Lemma 3 by dept-first-search of all possible variants, one gets a deterministic algorithm performing matching in polynomial time and exponential space.

Finally, the same argument can be applied for the case of non-deterministic and non-confluent SN P systems. In this case the simulating machine (RAM) must further keep trace of non-deterministic choices of rules in each step and try systematically all the possibilities. This requires obviously a polynomial space with respect to s and t.

5 Relation to standard complexity classes

For the first result we recall the NP-complete problem Subset Sum:

Problem 1. NAME: Subset Sum

INSTANCE: a (multi)set $V = \{v_1, v_2, \ldots, v_n\}$ of positive integers, and a positive integer S.
QUESTION: is there a sub(multi)set $B \subseteq V$ such that $\sum_{b \in B} = S$?

The usual agreed instance size of Subset Sum is $\Theta(n \log K)$, where $K = \max\{v_1, \ldots, v_n, S\}$.

Theorem 3. *The problem* Subset Sum *is solvable in semi-uniform way by a uniform family of deterministic recognizer SN P systems in constant time, where each member of the family solving an instance of size* $n \log K$ *has a single-neuron and a description of size* $\mathcal{O}(n \log K)$.

Proof. Consider the SN P system described in [16], Proposition 1 at p. 242–243. Given an instance of the Subset Sum problem defined ibidem, the construction of the SN P system is clearly polynomially uniform by Turing machine, the size of the description of the system is $\mathcal{O}(n \log K)$ (i.e., equivalent to the size of the instance), the system is deterministic and it solves the given instance in a single step.

Note: The above theorem remains valid only in the case of succinct representation of spikes and regular expressions. Otherwise the size of the SN P system would be nK, i.e., exponentially greater than the size of the instance.

The following lemma is due to [9] where detailed constructions of acyclic "neural" modules simulating gates with unbounded fan-in can be found.

Lemma 4. *SN P systems with regular expressions of the form* a^n, $n \geq 1$ *and without delays can simulate logic gates AND, OR, XOR with unbounded fan-in and fan-out in constant time and space.*

Theorem 4. *For an arbitrary polynomial* T, $\mathbf{CRCW}(poly(n), T(n)) \subseteq \mathbf{SN}_{-reg,-del}(T(n))$

Proof. By Lemma 4, $\mathbf{UCKT}(poly(n), T(n)) \subseteq \mathbf{SN}_{-reg,-del}(T(n))$ since a polynomially-sized circuit can be simulated in linear time a SN P system which can clearly be constructed in polynomial time by a Turing machine, and hence is a member of a uniform family. Then by Lemma 2, $\mathbf{CRCW}(poly(n), T(n)) \subseteq \mathbf{UCKT}(poly(n), T(n))$ for a polynomial T and, furthermore, the family of circuit is logspace-uniform, hence also polynomial time uniform by Turing machines. The whole construction of the simulating SN P system is therefore polynomially uniform and the statement of the theorem follows.

Note that the above result would hold for acyclic SN P systems as well since the neural modules referred to in Lemma 4 are acyclic. Since SN P systems allow for cyclic architectures, one might ask whether this fact could not improve the above result. Unlike acyclic circuits, SN P systems could simulate many steps of a CRCW PRAM program by the same computing modules. However, the size of the resulting SN P system would still increase with $P(n)$ and $T(n)$ as the number of bits processed by each RAM unit is bounded by $O(T(n))$ and the size of the SN P system increases accordingly. Hence in is interesting to observe that the computational power of uniform families of cyclic and acyclic SN P systems *differs only polynomially.* Solving the same problem, cyclic systems would have their size bounded by a polynomial of a lower degree.

The statement of the above theorem was partially foretold already in [10] where "neural" circuits of SN P systems performing arithmetics are presented. In that paper, however, these operations are performed sequentially, bit-by-bit, unlike in standard binary computers where all bits are processed in parallel.

Corollary 1. $\mathbf{P} = \mathbf{PSN}_{ssnf} = \mathbf{PSN}^*_{ssnf}$

Proof. Observe that $\mathbf{CRCW}(poly(n), poly(n)) = \mathbf{P}$, then the statement follows by Theorem 4 and 1.

Theorem 5. $\mathbf{NP} \subseteq \mathbf{PSN}^* \subseteq \mathbf{PSPACE}$

Proof. Follows by Theorems 2 and 3. Note that, thanks to the power of general regular expressions, there is no difference between constant and polynomial time.

We do not know whether for families of recognizer SN P systems also uniform solutions to NP-complete problems would be possible, allowed by some kind of "universal regular expression" for a given size of instances.

6 Beyond P and NP

Our first focus in this section is devoted to *non-uniform* families of SN P systems. This means that an unlimited number of resources and even a super-Turing computing devices can be used to construct members of the family. Clearly, without any further limitations, such families could solve any decision problem including undecidable ones. Therefore certain limitations are imposed especially on the size of members of the family. It is known that, to characterize the computing power of such families, the Turing machine with advice function is a useful tool. Informally, an advice is a binary string which is given to the Turing machine solving a decision problem in addition to its input. The advice is the same for all inputs of identical size, hence it depends only on the size of the input. It contains (possibly non-computable) correct information which can help to adopt a decision about the input.

Definition 7. *Let $f : N \to N$. An $f(n)$-advice sequence is a sequence of binary strings $A = (a_1, a_2, \ldots)$, where $|a_n| \leq f(n)$. For a language $B \subseteq \{0, 1, \#\}^*$ let $B@A = \{x| \ x\#a_{|x|} \in B\}$. Let $\mathbf{P}/f(n) = \{B@A : B \in P$ and A is an $f(n)$-advice sequence$\}$. Let $\mathbf{P}/\text{poly} = \bigcup_k \mathbf{P}/n^k$.*

Theorem 6. *Non-uniform families of deterministic recognizer SN P systems of polynomial size with regular expressions in single-star normal form can compute in polynomial time exactly the class of problems* $\mathbf{P}/\text{poly}$.

Proof. By Theorem 1, any polynomial-time restricted computations of a SN P system of a polynomial size with regular expressions in single-star normal form can be simulated by RAM (and hence also by Turing machine) in polynomial time. Furthermore, by Theorem 4 in [24], each computation of such a Turing machine can be simulated by a polynomial size circuit. Finally, by Theorem 2.2 in [2], each problem with polynomial circuit complexity is in $\mathbf{P}/\text{poly}$.

The converse inclusion follows by Lemma 4 which shows that each circuit of polynomial size can be simulated by a SN P system of an equivalent size with simple regular expressions.

Note that uniform solutions are necessary because they form a part of the standard definition of sets recognized by circuits [2]. If we thought about semi-uniform solutions, then for each instance we could have a special SN P system, and since their family is non-uniform, it could solve in constant time *any* problem, even non-decidable, by pre-computing the result.

Note also that, even if the sets recognized by the mentioned families of SN P systems are also recognized by a Turing machine in polynomial time, this does *not* imply that these sets are in $\mathbf{P}$ since these machines were derived from SN P systems constructed in a non-uniform way and hence they can contain an advice of polynomial size.

Finally we briefly mention SN P systems with pre-computed resources. The basic idea is that we start with an exponentially large number of inactive neurons, which will be subsequently activated and used during the computation. These neurons are arranged into a simple regular structure in the sense that a prototype neuron and the interconnection pattern is computed in polynomial time and then simply replicated [9]. It has been shown [13] that such SN P systems can solve in linear time **PSPACE**-complete problems QSAT and Q3SAT.

Conjecture 1. If a *deterministic* SN P system can activate an *exponential number of neurons* in a polynomial time, the resulting machine is equivalent to the parallel RAM, a standard Second Machine Class model, whose polynomial time-bounded computations characterize **PSPACE**.

Proof. (Sketch) The statement follows by Theorem 1, Theorem 4 and by the construction of CRCW PRAM as described, e.g., in [28].

Similar arguments indicate that, if a *non-deterministic* SN P system can activate an *exponential number of neurons* in a polynomial time, the resulting machine may prove equivalent to the P-RAM [7], a computing model whose non-deterministic polynomial time-bounded computations characterize the class **NEXPTIME**. These open problems are the next to be addressed in the future research.

7 Conclusion

In this contribution we tried to generalize results concerning complexity issues of SN P systems published previously in a sequence of papers. We have defined (uniform) families of recognizer SN P systems which should still be more elaborated in future work. Main results can be summarized as follows:

1. Polynomially uniform families of recognizer SN P systems with regular expressions in *single-star normal form* characterize by their polynomial time-bounded computations the class **P**.
2. Polynomially uniform families of recognizer SN P systems with *general regular expressions* can in polynomial time solve a class of problems bounded between **NP** and **PSPACE**.
3. If a confluent or deterministic SN P system can activate an *exponential number of neurons* in a polynomial time, it is probably computationally equivalent to parallel RAM, a standard Second Machine Class model whose computations in polynomial time characterize **PSPACE**.
4. *Non-uniform* families of SN P systems of *polynomial size* with regular expressions in *single-star normal form* characterize by their *uniform solutions* in polynomial time the class **P**/poly.

Rather surprisingly, it turned out that there is no substantial difference in power of uniform families cyclic and acyclic SN P systems. The use of single-star normal form for regular expressions to demonstrate the borderline between SN P systems able to solve tractable and intractable problems is also interesting.

Acknowledgements

Authors are grateful to Mario Pérez-Jiménez for suggestions about definitions of complexity classes for SN P systems. The research was partially supported by the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2009–14421, by the program I3, by the Comunidad de Madrid (grant No. CCG06-UPM/TIC-0386 to the LIA research group), and by the Silesian University in Opava, grant No. SGS/4/2010.

References

1. S. Andrei, S.V. Cavadini, and W.-N. Chin. A new algorithm for regularizing one-letter context-free grammars. *Theoretical Computer Science*, 306:113–122, 2003.

2. R.B. Boppana and M. Sipser. The complexity of finite functions. In van Leeuwen [29], pages 757–804.
3. Cristian S. Calude, José Félix Costa, Rudolf Freund, Marion Oswald, and Grzegorz Rozenberg, editors. *Unconventional Computing, 7th International Conference, UC 2008, Vienna, Austria, August 25-28, 2008. Proceedings*, volume 5204 of *Lecture Notes in Computer Science*. Springer, 2008.
4. D. Díaz-Pernil et al., editor. *Proceedings of Sixth Brainstorming Week on Membrane Computing*, Sevilla, 2008. Fénix Editora.
5. George Eleftherakis, Petros Kefalas, and Gheorghe Paun, editors. *Proceedings of the 8th International Workshop on Membrane Computing*. SEERC, 2007.
6. George Eleftherakis, Petros Kefalas, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors. *Membrane Computing, 8th International Workshop, WMC 2007*, volume 4860 of *Lecture Notes in Computer Science*. Springer, 2007.
7. S. Fortune and J. Wyllie. Parallelism in random access machines. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118, New York, NY, USA, 1978. ACM.
8. L.M. Goldschlager. A universal interconnection pattern for parallel computers. *J. ACM*, 29(4):1073–1086, 1982.
9. M.A. Gutiérrez-Naranjo and A. Leporati. Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. In D. Díaz-Pernil et al. [4], pages 193–210.
10. M.A. Gutiérrez-Naranjo and A. Leporati. Performing arithmetic operations with spiking neural P systems. In Martínez-del-Amor et al. [19], pages 181–198. Volume 1.
11. M.A. Gutiérrez-Naranjo, G. Păun, A. Romero-Jiménez, and A. Riscos-Núñez, editors. *Proceedings of Fifth Brainstorming Week on Membrane Computing*, Sevilla, 2007. Fénix Editora.
12. M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2–3):279–308, 2006.
13. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, and X. Zhang. Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science*, In Press, 2010.
14. R.M. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In van Leeuwen [29], pages 869–942.
15. A. Leporati, G. Mauri, C. Zandron, G. Păun, and M.J. Pérez-Jiménez. Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, 8(4):681–702, 2009.
16. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. On the computational power of spiking neural P systems. In Gutiérrez-Naranjo et al. [11], pages 227–245.
17. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. Solving numerical NP-complete problems with spiking neural P systems. In Eleftherakis et al. [6], pages 336–352.
18. W. Maass and C. Bishop, editors. *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.
19. M.A. Martínez-del-Amor, E.F. Orejuela-Pinedo, G. Păun, I. Pérez-Hurtado, and A. Riscos-Núñez, editors. *Seventh Brainstorming Week on Membrane Computing*, Sevilla, 2009. Fénix Editora.
20. T. Neary. On the computational complexity of spiking neural P systems. In Calude et al. [3], pages 189–205.
21. The P Systems Web Page. `http://ppage.psystems.eu/`. [cit. 2009-12-29].

22. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, Oxford, 2009.
23. M.J. Pérez-Jiménez. A computational complexity theory in membrane computing. In Păun et al. [27], pages 125–148.
24. N. Pippenger and M.J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.
25. A.E. Porreca, A. Leporati, G. Mauri, and C. Zandron. Introducing a space complexity measure for P systems. *Int. J. of Computers, Communications & Control*, IV(4):301–310, 2009.
26. G. Păun. *Membrane Computing – An Introduction.* Springer, Berlin, 2002.
27. G. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors. *Membrane Computing, 10th International Workshop, WMC 2009*, volume 5957 of *Lecture Notes in Computer Science*, Berlin, 2010. Springer.
28. L. Stockmeyer and U. Vishkinj. Simulation of parallel random access machines by circuits. *SIAM J. Comput.*, 13 (2):409–422, 1984.
29. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity. Elsevier, Amsterdam, 1990.

On Catalytic P Systems with One Catalyst

Dragoş Sburlan

Faculty of Mathematics and Informatics
Ovidius University of Constanta, Romania
dsburlan@univ-ovidius.ro

Summary. In this paper we address the possibility of studying the computational capabilities of catalytic P systems with one catalyst by the means of iterated finite state transducers. We also give a normal form for catalytic P systems.

1 Introduction

P systems are a computational model introduced by G. Păun in [4]. One of the basic variant considered there was P systems with catalysts and priorities; these systems where shown to be computationally universal. In [2], Sosík and Freund proved that priorities among the rules can be discarded from the model without any loss of computational power. Moreover, it was shown that for extended P systems only one membrane and two catalysts are enough for reaching computational universality. However, the computational power for P systems with only one catalyst was not established. The present paper characterize these systems in terms of iterated finite state transducers hence it converts an open problem from P system framework to an open problem from string rewriting theory. Additionally, a normal form for catalytic P systems is presented.

2 Preliminaries

We assume the reader is acquainted with the basic notions and notations from the formal language theory (see [3] for more details). Here we only recall the definitions and the results which are useful for the present work.

If FL is a family of languages, then NFL denotes the family of length sets of languages in FL. We denote by *REG*, *CF*, *REC*, and *RE* the family of regular, context-free, recursive, and recursively enumerable languages, respectively. It is known that $NREG = NCF \subsetneq NREC \subsetneq NRE$.

2.1 Iterated Finite State Transducers

An *iterated (finite state) sequential transducer* (IFT) is a construct $\gamma = (K, V, q_0, a_0, F, P)$, where K is a finite set of *states*, V is a finite set of symbols (the *alphabet* of γ), $K \cap V = \emptyset$, $q_0 \in K$ is the *initial state*, $a_0 \in V$ is the *starting symbol*, $F \subseteq K$ is the set of *final states*, and P is a finite set of *transition rules* of the form $qa \to xp$, for $q, p \in K$, $a \in V$, and $x \in V^*$.

For $q, p \in K$ and $u, v, x \in V^*$, $a \in V$, a *direct transition* step of γ is defined as $uqav \vdash uxpv$ if and only if $qa \to xp \in P$. The reflexive and transitive closure of the relation $\vdash$ is denoted by $\vdash^*$. In general, for $\alpha, \beta \in V^*$ we say that α *derives* into β and we write $\alpha \Longrightarrow \beta$, if and only if $q_0\alpha \vdash^* \beta p$ for some $p \in K$. By $\Longrightarrow^*$ we denote the reflexive transitive closure of $\Longrightarrow$. If $q_0\alpha \vdash^* \beta p$ such that $p \in F$, then we write $\alpha \Longrightarrow_f \beta$.

The language generated by γ is $L(\gamma) = \{\beta \in V \mid a_0 \Longrightarrow^* \alpha \Longrightarrow_f \beta,\ \text{for some } \alpha \in V^*\}$.

If for each pair $(q, a) \in K \times V$, there is at most one transition rule $qa \vdash xp \in P$, then γ is called *deterministic* (otherwise, it is called *nondeterministic*). The family of languages generated by nondeterministic IFTs with at most $n \geq 1$ states is denoted by IFT_n. It is known from [1] that $CF \subset IFT_2 \subseteq IFT_3 \subseteq IFT_4 = RE$. Moreover, there are non-semilinear languages belonging to IFT_2, and there are non-recursive languages belonging to IFT_3. Consequently, if we denote by $NIFT_n$, $n \geq 1$, the family of length sets of languages from IFT_n, then we have that $NREG = NCF \subsetneq NIFT_2 \subseteq NIFT_3 \subseteq NIFT_4 = NRE$.

2.2 Membrane Systems

A *catalytic P system* of degree $m \geq 1$ is a construct

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$$

where

- O is an alphabet of *objects*;
- $C \subseteq O$ is the set of *catalysts*;
- μ is a hierarchical tree structure of $m \geq 1$ uniquely labelled *membranes* (which delimit the regions of Π); typically, the set of labels is $\{1, \ldots, m\}$;
- $w_i \in O^*$, for $1 \leq i \leq m$, are the multisets of objects initially present in the m regions of μ;
- R_i, $1 \leq i \leq m$, are finite sets of *evolution rules*; these rules can be *non-cooperative* $a \to v$ or *catalytic* $ca \to cv$, where $a \in O \setminus C$, $v \in ((O \setminus C) \times \{here, out, in\})^*$, and $c \in C$;
- $i_0 \in \{1, \ldots, m\}$ is the label of the *output region* of Π.

A *configuration* of Π is a vector $C = (\alpha_1, \ldots, \alpha_m)$, where $\alpha_i \in O^*$, $1 \leq i \leq m$, is a multiset of objects present in the region i of Π. The vector $C_0 = (w_1, \ldots, w_m)$ is the *initial configuration* of Π. Starting from the initial configuration and always applying in all membranes a maximal multiset of evolution rules in parallel, one gets a sequence of consecutive configurations. By $\Rightarrow$ is denoted the *transition* between two consecutive configurations. A sequence (finite or infinite) of transitions starting from C_0 represents a *computation* of Π. A computation of Π is a halting one if no rules can be applied to the last configuration (the *halting configuration*). The result of a halting computation is the number of objects from O contained in the output region i_0, in the halting configuration. A non-halting computation yields no result. By collecting the results of all possible halting computations of a given P system Π, one gets $N(\Pi)$ – the set of all natural numbers generated by Π. The family of all sets of numbers computed by catalytic P systems with at most m membranes and k catalysts is denoted by $NOP_m(cat_k)$. The above definition can be relaxed such that in a halting configuration one counts only the symbols from a given alphabet $\Sigma \subseteq O$. In particular, one can consider $\Sigma = O \setminus C$; correspondingly, the family of all sets of numbers computed by such particular P systems will be denoted by $NO_{-C}P_m(cat_k)$.

It is known (see [7], for instance) that $NO_{-C}P_m(cat_k) = NO_{-C}P_1(cat_k)$. Moreover, in [2] it is shown that $NO_{-C}P_1(cat_2) = NRE$.

3 A Normal Form for P Systems with Catalysts

The following result states that any catalytic P system is equivalent with a catalytic P system having a restriction on the form of the rules.

Theorem 1. *For any P system Π with catalysts there exists an equivalent P system $\overline{\Pi}$ with one region and whose rules are of the form $a \to \alpha$, with $|\alpha| \le 2$, or $ca \to c\beta$, with $|\beta| \le 1$.*

Proof. As we already stated in Section 2.2, for any P system with catalysts and $n > 1$ membranes one can construct an equivalent P system with the same number of catalysts and one membrane. Consequently, without loss of generality, we might assume that Π has only one membrane, that is $\Pi = (O, C, [\,]_1, w_1, R_1, i_0)$.

Let $O \setminus C = \{a_1, a_2, \ldots, a_p\}$ and let $m = max\{|\alpha| \mid a \to \alpha \in R_1$ or $ca \to c\alpha \in R_1\}$. In addition, assume for our convenience that the rules of Π are labeled in an unique manner with numbers from the set $\{1, \ldots, card(R_1)\}$.

Then one can construct an equivalent P system $\overline{\Pi} = (\overline{O}, C, [\ ]_1, w_1, \overline{R_1}, i_0)$ where

$$\begin{aligned}\overline{O} = O &\cup \{a_{(i,j)} \mid 1 \le i \le p, 1 \le j \le m\} \\ &\cup \{X_{(i,j)} \mid i : a \to \alpha_i \in R_1, 1 \le j \le m-2\}.\end{aligned}$$

The set $\overline{R_1}$ is defined as follows (for the simplicity of the explanations, we will only consider the rules in $\overline{R_1}$ that are useful for simulating a non-cooperative rule from R_1; the rules corresponding to a catalytic rule are defined similarly, therefore we will not present them here). Let $i : a \to a_{j_1} a_{j_2} \ldots a_{j_k} \in R_1$ and let $m - k = t$. Then we add to $\overline{R_1}$ the rules:

$$\begin{aligned} a &\to X_{(i,1)} \\ X_{(i,1)} &\to X_{(i,2)} \\ &\dots \\ X_{(i,t-1)} &\to X_{(i,t)} \end{aligned} \tag{1}$$

$$\begin{aligned} X_{(i,t)} &\to a_{(j_1,k-1)}X_{(i,t+1)} \\ X_{(i,t+1)} &\to a_{(j_2,k-2)}X_{(i,t+2)} \\ &\dots \\ X_{(i,t+k-3)} &\to a_{(j_{k-2},2)}X_{(i,t+k-2)} \\ X_{(i,t+k-2)} &\to a_{(j_{k-1},1)}a_{(j_k,1)} \end{aligned} \tag{2}$$

$$\begin{aligned} a_{(i,m)} &\to a_{(i,m-1)} \\ a_{(i,m-1)} &\to a_{(i,m-2)} \\ &\dots \\ a_{(i,1)} &\to a_i \end{aligned} \tag{3}$$

The proof is based on the existence of the universal global clock that governs the functioning of the P system (the clock marks equal time units for the whole system, hence synchronization is possible). While trying to simulate the application of an arbitrary non-cooperative rule with several rules of type $a \to \alpha$, with $|\alpha| \leq 2$, one has to accomplish two conditions. Firstly, one has to guarantee that all the objects from α will eventually be produced. Secondly, these objects must be produced at the "proper" time: all of them in the same moment (a local synchronization) and according with the simulation of other rules that were started at the same time with $a \to \alpha$ (a global synchronization).

Consequently, the rules presented above are grouped according with their function in the simulation. The first group represents a set of "delaying" rules (they are used while simulating the rules with a shorter right hand side in order to synchronize their executions with those that have the longest right hand side). These rules are "chained", hence, staring from an object a, an object $X_{(i,t)}$ is produced in exactly t computational steps. The second group is responsible for producing in consecutive computational steps the objects $a_{(j_1,k-1)}, a_{(j_2,k-2)}, \dots, a_{(j_{k-1},1)}, a_{(j_k,1)}$ (in order of their production, the last two being produced in the same time). For an object $a_{(i,l)}$ in this sequence, the index l represents the number of computational steps that $\overline{\Pi}$ will perform, starting from its production and until the object a_i is produced (see the third group of rules). Finally, one can remark that the objects $a_{j_1}, a_{j_2}, \dots, a_{j_k}$ are produced in the same computational step by $\overline{\Pi}$ (while simulating the rule $i : a \to a_{j_1}a_{j_2}\dots a_{j_k} \in R_1$). Moreover, all the other rules from Π that stated at the same moment as $i : a \to a_{j_1}a_{j_2}\dots a_{j_k}$, are simulated in the same manner by $\overline{\Pi}$ and their output is produced in the same computational step as mentioned above. Consequently $\overline{\Pi}$ correctly simulates any computation of Π, hence the theorem holds true.

4 Catalytic P Systems with One Catalyst and IFTs

In what follows we prove that the family of sets of numbers computed by catalytic P systems with only one catalyst is included in the family of the length sets of the languages generated by iterated finite state transducers with at most 3 states.

Theorem 2. $NIFT_3 \supseteq NOP_1(cat_1)$.

Proof. Given an arbitrary catalytic P system $\Pi = (O, C, w_1, R_1, i_1)$ such that $C = \{c\}$, then one can construct an iterated finite transducer $\gamma = (K, V, q_0, a_0, F, P)$ which simulates Π as follows.

Without loss of generality we assume that the initial configuration of Π is $w_1 = ca_0$.

Let $w = a_1a_2 \dots a_m$ be a string. We denote by

$$\mathrm{Perm}(w) = \{a_{i_1}a_{i_2} \dots a_{i_m} \mid 1 \le i_j \le m, 1 \le j \le m, \text{ with } i_j \ne i_l, 1 \le j, l \le m\}$$

the set of all permutations of string w, i.e., the set of all strings that can be obtained from w by changing the order of symbols.

In addition, let us consider the following sets of objects from O:
$X = \{A \in O \mid (\exists)\ A \to \alpha \in R_1 \text{ and } (\nexists)\ cA \to c\beta \in R_1\}$;
$Y = \{A \in O \mid (\exists)\ A \to \alpha \in R_1 \text{ and } cA \to c\beta \in R_1\}$;
$Z = \{A \in O \mid (\exists)\ cA \to c\alpha \in R_1 \text{ and } (\nexists)\ A \to \beta \in R_1\}$;
$T = \{A \in O \mid (\nexists)\ A \to \alpha \in R_1 \text{ and } (\nexists)\ cA \to c\beta \in R_1\}$.
One can remark that $O = X \cup Y \cup Z \cup T \cup \{c\}$.
Based on the above settings the IFT γ is defined as follows:

$$K = \{q_0, q_1, q_2\},$$
$$V = O \setminus \{c\},$$
$$F = \{q_0\},$$

and the set of rules P is constructed in the following manner:

• for any $a \in T$ we add to P the rule $q_0a \to aq_0$;

• for any $a \in X \cup Y$ and $a \to \alpha \in R_1$ we add to P the rules $q_0a \to \overline{\alpha}q_1$, where $\overline{\alpha} \in \mathrm{Perm}(\alpha)$;

• for any $a \in T$ we add to P the rule $q_1a \to aq_1$;

• for any $a \in X \cup Y$ and $a \to \alpha \in R_1$ we add to P the rules $q_1a \to \overline{\alpha}q_1$, where $\overline{\alpha} \in \mathrm{Perm}(\alpha)$;

• for any $a \in Y \cup Z$ and $ca \to c\alpha \in R_1$ we add to P the rules $q_1a \to \overline{\alpha}q_2$, where $\overline{\alpha} \in \mathrm{Perm}(\alpha)$;

• for any $a \in T \cup Z$ we add to P the rule $q_2a \to aq_2$;

• for any $a \in X \cup Y$ and $a \to \alpha \in R_1$ we add to P the rules $q_2a \to \overline{\alpha}q_2$, where $\overline{\alpha} \in \mathrm{Perm}(\alpha)$;

• for any $a \in Y \cup Z$ and $ca \to c\alpha \in R_1$ we add to P the rules $q_0 a \to \overline{\alpha} q_2$, where $\overline{\alpha} \in \mathrm{Perm}(\alpha)$.

The construction was designed such that each string processed by γ during its computation will correspond to a configuration of Π. Moreover, one iteration of γ simulates the maximal parallel applications of the rules of Π.

If the current string (say w) processed by the IFT is composed only by the symbols from T, then γ remains in $q_0 \in F$ and stops, accepting the string. This situation corresponds to the halting configuration of Π (that is, Π contains in its region the multiset cw and no rules can be further applied).

In case w contains symbols form $X \cup Y \cup Z$, then γ starts the simulation of the maximal parallel applications of the rules of Π. Since γ processes strings at each iteration, then the simulation of Π has to accomplish the following task: all the symbols which are the subject of a rule of Π have to be processed also by γ. Recall that γ processes strings and in these strings there might be symbols from T (which are not the subject of any rule) in any position. Consequently, one has be sure that any symbol in a configuration of Π that is a subject of a rule (non-cooperative or catalytic) has to have the opportunity to be rewritten in the corresponding string processed by γ (by the corresponding rule). This is why, γ uses the rules $q_i a \to a q_i$ for $q_i \in Q$, $1 \leq i \leq 3$, and $a \in T$ (that is, while processing the string, γ "skips" all the symbols that are not the subject of any rule).

In one iteration of γ one can apply at most once a rule corresponding to a catalytic rule of Π (recall that the P system functioning semantics define such behaviour). More precisely, assuming that w is the current processed string, we have

- either γ is in state q_0 and executes a rule of type $q_0 a \to \overline{\alpha} q_2$ for $q_1, q_2 \in Q$, $a \in Y \cup Z$, $ca \to c\alpha \in R_1$, and $\overline{\alpha} \in \mathrm{Perm}(\alpha)$. This situation occurs when γ processes $w = w_1 a w_2$, $w_1 \in T^*$, and $w_2 \in (X \cup Y \cup Z \cup T)^*$ (w has the prefix w_1 composed only by symbols from T, followed by the symbol $a \in Y \cup Z$ that triggers the simulation of the catalytic rule; the symbols from w_2 that belong to $X \cup Y$ will trigger only the simulation of the non-cooperative rules).
- either γ is in state q_1 and executes a rule of type $q_1 a \to \overline{\alpha} q_2$ for $q_1, q_2 \in Q$, $a \in Y \cup Z$, $ca \to c\alpha \in R_1$, and $\overline{\alpha} \in \mathrm{Perm}(\alpha)$). This situation occurs when γ processes $w = w_1 a w_2$, where w_1 is described by the regular expression $T^*(X|Y)(X|Y|T)^*$, $w_2 \in (X \cup Y \cup Z \cup T)^*$ (the symbols from w_1 and w_2 that belong to $X \cup Y$ will trigger only the simulation of the non-cooperative rules) .

One can also remark that if a configuration w of Π contains at least one object $a \in Z$, then in the current computational step a catalytic rule will be executed (because of the maximal parallel applications of the rules); in contrary, if w does not contain any symbol $a \in Z$ then it is not guaranteed that a catalytic rule will be executed (even if w contains symbols from Y, then, because of the nondeterminism, it might happen that all the rules selected for application are non-cooperative). On the other hand, γ simulates Π by processing strings (hence the order of symbols is precisely defined). The design of γ guarantees that, if applicable, a rule corresponding to a catalytic rule of Π is executed at most once. The only issue that could appear regards the presence of multiple symbols from $Y \cup Z$ in the current string processed by γ (in order to perform a correct simulation, one has to be sure that any of these symbols has a "chance" to be rewritten). This is why for any rule $a \to \alpha \in R_1$ or $ca \to c\alpha \in R_1$, the IFT γ will use for the simulation a set of rules of the type $qa \to p\mathrm{Perm}(\alpha)$.

Based on the above theorem, the following result holds true.

Corollary 1. *If* $NIFT_3 \subset NRE$ *then* $NOP_1(cat_1) \subset NRE$

5 Conclusions

In this paper we gave a normal-form theorem for catalytic P systems. We also investigated the relation between P systems with one catalyst and iterated finite transducers. This last topic is of a particular interest because it converts an open problem from the P system framework to an open problem from the string rewriting theory. In addition, the simplicity of the construction gives hopes for solving an open problem stated from the introduction of P systems.

Acknowledgements

The work of the author was supported by the CNCSIS IDEI-PCE grant, no. 551/2009, Romanian Ministry of Education, Research and Innovation.

References

1. Bordihn H., Fernau H., Holzer M., Manca V., Martin-Vide C., Iterated Sequential Transducers as Language Generating Devices, *Theoretical Computer Science*, 369, 1 (2006), pp. 67–81.
2. Freund R., Kari L., Oswald M., Sosik P., Computationally Universal P Systems Without Priorities: Two Catalysts Are Sufficient, *Theoretical Computer Science*, 330, 2 (2005), pp. 251–266.
3. Rozenberg G., Salomaa A., eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.
4. Păun G., Computing with Membranes: an Introduction. *Bulletin EATCS*, 67 (1999), pp. 139–152.
5. Păun G., *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
6. Păun G., Rozenberg G., Salomaa A., eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press Inc., New York, 2010.
7. Sburlan D., Further Results on P Systems with Promoters/Inhibitors, *International Journal of Foundations of Computer Science*, 17 (2006), pp. 205–221.

"Dogmatic" P Systems*

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
jsempere@dsic.upv.es

Summary. In this work we propose a variant of P systems based on the Central Dogma of Molecular Biology which establishes the transformation of DNA strands into protein products by applying different string transformation such as transductions and transcriptions. We introduce a new kind of worm object rules to carry out transducion operations. Finally, we establish the universality of the proposed model by simulating Iterated finite state sequential transducers (IFTs).

1 Introduction

P systems [15] were introduced as a computational model inspired by the information and biochemical product processing of living cells through the use of membrane communication. In most of the works about P systems, information is represented as multisets of symbol/objects which can interact and evolve according to predefined rules. Nevertheless, the use of strings to represent the information and the use of rules to transform strings instead of multisets of objects have always been present in the literature of this scientific area. So, in his mostly referred book [15], Gh. Păun overviews the use of string rules in P systems. Different variants of string-based P systems have been proposed along the time. We can mention *rewriting P systems* [11], referred as membrane systems with *worm objects* [2] in the case of genomic operations, *insertion-deletion P systems* [6] and *splicing P systems* [14], among others. Observe that most of these models have been used for language generation [12]. In [5, 7], the proposal of *hybrid P systems* introduces the use of contextual rules and Chomsky rules to achieve universality by generating all the recursively enumerable languages. Recently, in [13] a variant of P systems with worm objects and evolutionary based operations has been introduced to simulate Networks of Evolutionary Processors, hence to achieve universality.

In this work, we propose a variant of P systems with worm objects and a new kind of worm rules based on the central dogma of molecular biology which sets

* Work supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769

the framework to obtain protein products from DNA strands by applying, among others, transduction and transcription operations.

The structure of this work is as follows: In section 2 we introduce basic concepts and notation on formal language theory, iterated transductions, P systems and molecular biology related to the *Central Dogma*. Then, we will define the dogmatic rules in regions which transduce (fragments of) worm objects into (fragments of) worm objects. We will propose a simulation of iterated transductions with the new proposed model in order to achieve universality. Finally, we will outline future research related to this work.

2 Basic Concepts

We start by summarizing the notions used throughout this work. An *alphabet* is a finite and nonempty set of symbols. Any finite sequence of symbols from an alphabet V is called *word* or *string* over V. The set of all words over V is denoted by V^*. A *language* over the alphabet V is any subset of V^*.

A grammar is a construct $G = (N, \Sigma, P, S)$ where N and Σ are the alphabets of auxiliary and terminal symbols with $N \cap \Sigma = \emptyset$, $S \in N$ is the *axiom* of the grammar and P is a finite set of productions in the form $\alpha \to \beta$, where $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$ and $\beta \in (N \cup \Sigma)^*$. The language of the grammar is denoted by $L(G)$ and it is the set of terminal strings that can be obtained from S by applying symbol substitutions according to P. Formally, $w_1 \underset{G}{\Rightarrow} w_2$ if $w_1 = u\alpha v$, $w_2 = u\beta v$ and $\alpha \to \beta \in P$. We will denote by $\overset{*}{\underset{G}{\Rightarrow}}$ the reflexive and transitive closure of $\underset{G}{\Rightarrow}$. So, the language generated by G is defined by the set $L(G) = \{w \in \Sigma^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

Four larger families of languages generated by grammars can be defined: REG (regular), CF (context-free), CS (context-sensitive) and RE (recursively enumerable). The definition of these families comes from the restriction over the production forms in the grammar. The well known Chomsky's hierarchy establishes the inclusions $REG \subset CF \subset CS \subset RE$.

Iterated Transductions

In the following, we will introduce *Iterated finite state sequential transducers* (IFT) as it was defined in previous works ([1, 8, 10]).

An IFT is defined by the tuple $T = (Q, \Sigma, q_0, a_0, F, P)$, where Q is a finite set of *states*, Σ is an alphabet, $q_0 \in Q$ is an *initial state*, $a_0 \in \Sigma$ is a *starting symbol*, $F \subseteq Q$ is the set of *final states* and P is a finite set of *transduction rules* in the form (q, a, p, x) with $q, p \in Q$, $a \in \Sigma$ and $x \in \Sigma^*$ which we will write as $qa \to xp$. The transduction rule $qa \to xp$ means that if the finite control is in state q and it reads the symbol a then it changes to state p and writes x. We define a *direct transition step* as follows

$$uqav \vdash uwpv \text{ iff } qa \to wp \in P$$

The reflexive and transitive closure of $\vdash$ will be denoted by $\vdash^*$. We say that w derives x, and it will be denoted by $w \Longrightarrow x$, iff $q_0 w \vdash^* xp$, for $p \in Q$ (observe that p is any state in Q not necessarily final). We will denote the reflexive and transitive closure of $\Longrightarrow$ by $\Longrightarrow^*$. If in the previous derivation the process stops in a final state we will write $\overset{f}{\Longrightarrow}$ instead of $\Longrightarrow$. That is, $w \overset{f}{\Longrightarrow} x$, iff $q_0 w \vdash^* xp$, for $p \in F$. The language generated by T is defined as follows

$$L(T) = \{x \in \Sigma^* : a_0 \Longrightarrow^* w \overset{f}{\Longrightarrow} x, w \in \Sigma^*\}$$

We denote by IFT_n the family of languages generated by IFT with at most n states. The hierarchy of families in IFT_n has been completely explored, and it has been proved that it collapses at level four. We have the following results

Lemma 1.[10] $RE = IFT_4$; [1] $CS \subset IFT_3$; [10] $CF \subset IFT_2$.

In addition, IFTs have been related to the *computing by carving* paradigm [9] as a way to generate even non-recursively enumerable languages.

The Central Dogma of Molecular Biology

The *Central Dogma* of Molecular Biology is our source of inspiration for the variant of P system which we will propose later. We follow the ideas exposed in [4]. Mainly, the central dogma of molecular biology establishes a metaphor of how DNA strands in the living cell are transformed into protein products by means of information storage and transformation.

Mainly, a section of DNA (the gene) is transcribed to a molecule of messenger RNA and the mRNA is translated by the ribosome into a protein. In the eukaryotic organisms the mRNA molecule is processed, before translation, by splicing out certain subsequences called *introns*. The DNA is replicated before the transcription. The transcription is made by complementing the single DNA strand, and by substituting the thymine nucleotide by the uracil one in the RNA molecule. The translation from (spliced) mRNA to proteins is based on a mapping of nucleotide triplets called *codons* to amino acids with the help of transfer RNA (tRNA). Under a computer science point of view, the central dogma can be viewed as a sequence of well known operations over strings such as morphisms, transductions and splicing. The main ingredients that we will consider in the subsequent P system that we will propose are the followings:

- There are different processes in different regions. DNA duplication and DNA transcription to mRNA occurs in the nucleus of the cell, while mRNA translation to amino acids occurs in some cases in the endoplasmic reticulum with the membrane ribosome.
- There are different alphabet sizes and symbols involved in the operations. The DNA strands is a sequence of four different nucleotides: adenine (A), thymine (T), cytosine (C) and guanine (G), in the RNA the thymine (T) is substituted by the uracil (U), while the proteins are sequences over a twenty-letter alphabet (the amino acids)

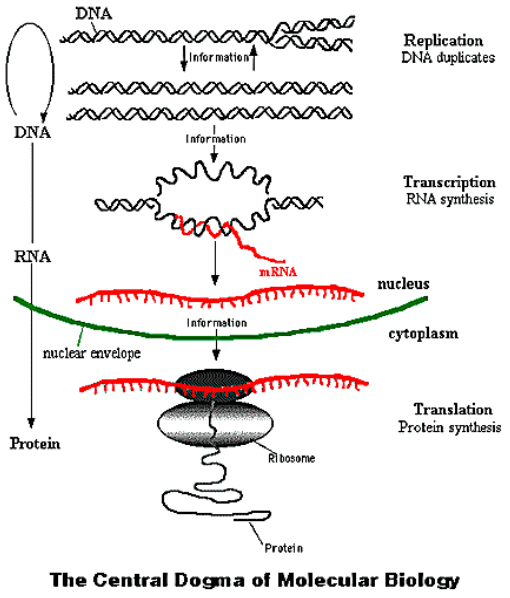


Fig. 1. The Central Dogma of Molecular Biology. (This picture has been taken from accessexcellence.org)

- Transcription and translation can be performed by alphabetic homomorphisms and finite transductions.
- There are different products at every stage which interacts into different regions. The DNA duplication, transcription and splicing needs the presence of different proteins and other molecular compounds. The proteins are the final product of the cycle DNA-RNA-protein.

3 Dogmatic P systems

In this section, we will propose a variant of P systems that work with worm objects in a transduction-like approach. First, we will introduce a new kind of region rules to work with.

A *dogmatic* rule is defined as follows

$$u : v_{pos} \longrightarrow w_{ad_1,ad_2,\cdots,ad_k}, \text{ where}$$

u, v are strings (worm objects), $pos \in \{l, r, *\}$ and for all $i : 1 \leq i \leq k$ $ad_i \in \{here, out, in_j\}$. The meaning is the following: Provided that there exist a worm object u in the region (we can omit the presence of u), all the worm objects with substring v at position pos (which means, rightmost one (r), leftmost one (l) or arbitrary position ($*$)) change substring v by w and send a copy of the new worm object at the regions defined by ad_i after eliminating the original worm object from the region.

Example 1. Let the region R have the rule r_1 defined as $eee : a_l \rightarrow bb_{here}$ and the worm objects eee and $abbcbaa$. Then after applying r_1 in the region, the worm objects are eee and $bbbbcbaa$.

If the rule r_1 is defined as $eee : a_r \rightarrow bb_{here}$, we obtain $abbcbabb$ as a new worm object. Finally, if the rule is defined as $eee : a_* \rightarrow bb_{here}$ then we obtain the set of new strings $\{bbbbcbaa, abbcbbba, abbcbabb\}$. Observe that, in this case, we have previously obtained three copies of the initial string before applying the rule.

The rule $a_l \rightarrow bb_{here}$ can be applied over baa and it obtains the new string $bbba$. Here, we have omitted the presence of an additional string and the rule changes the leftmost appearance of a symbol a. $\square$

The addressing label in_j, can be directly applied to contiguous regions at the same level. That is, if there exist regions j and i inside the same region, then a rule at region i can send worm objects to region j directly.

We can observe that the dogmatic rules capture the following aspects from the Central Dogma of Molecular Biology:

- The rules transform parts of a string into a new substring as in transcription and transduction.
- The rules make copies of the target string before transformation as in DNA replication.
- The rules need the presence of other objects to be applied.
- The rules can address contiguous regions (i.e. RNA moving from nucleus to ribosomes).

Now, we will define a *Dogmatic P system*[2] as the following construct

$$\Pi = (V, \mu, A_1, \cdots, A_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0), \text{ where:}$$

- V is an alphabet
- μ is a membrane structure consisting of m membranes
- A_i, $1 \leq i \leq m$ is a finite set of strings associated with the region i (the *axioms*)
- R_i, $1 \leq i \leq m$ is a finite set of *dogmatic rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals to ∞ the output is read outside the system).

[2] Different acronyms were candidates for naming Dogmatic P systems. Among others, *dP* systems were considered but it was previously used by other authors in a different context. Another acronym was *dogP* but the author thinks that, in such a case, catalyzers will never be used in this context given that "*dogs*" and "*cats*" could not cooperate and living in the same regions. We leave open the search for a good acronym for the proposed Dogmatic *P* systems.

Initially, the system holds the set of axioms at every region. Then, in a fully parallel manner all the rules are applied over the strings defined at every region. The system halts whenever no rule can be applied at any region.

The language generated by Π is the set of worm objects collected at region i_0. In the case that $i_0 = \infty$, the language is collected in *external mode* as the set of strings in the environment. The language generated by Π is denoted by $L(\Pi)$. Observe that if the language is infinite then the system will never halt so it will add new worm objects to the output region or the environment.

Observe that this proposal is different from [3] where the authors propose a membrane system framework with symport/antyport rules to perform different types of transductions. In that work the proposed system operates with strings by taking every symbol of the input string of the environment (outside the membrane system) and putting every symbol of the transduced string in the environment. Here, we will avoid symport/antyport rules and we will work with strings in a worm object approach.

4 A Simulation of Iterated Transductions by Dogmatic P Systems

In this section, we will show a simulation of IFTs with n states by dogmatic P Systems. Our approach will use n regions inside the skin one in order to simulate the n states of the IFT. The transitions of the IFT will be simulated by using the direct address in_j. We will need to mark some symbols in order to carry out the transduction from left to right. In addition, we will use different alphabets to avoid a wrong application of the transduction rules at different symbols, and to prevent that the simulation goes on even if the IFT cannot carry out a complete transduction.

Let $T = (Q, \Sigma, q_0, a_0, F, P)$ be an IFT with $Q = \{q_0, \cdots, q_n\}$. Then, we propose the following dogmatic P system

$$\Pi = (V, \mu, A, A_0, \cdots, A_n, (R, \rho), (R_0, \rho_0), \cdots, (R_n, \rho_n), \infty), \text{ where}$$

- $V = \Sigma \cup \hat{\Sigma} \cup \breve{\Sigma} \cup \{\#\}$, where $\hat{\Sigma} = \{\hat{a} : a \in \Sigma\}$ and $\breve{\Sigma} = \{\breve{a} : a \in \Sigma\}$
- $\mu = [[_0]_0, \cdots, [_n]_n]$ (we have omitted a label for the skin region).
- $A_0 = \{\#a_0\}$, $A = \emptyset$, and for all $i : 1 \leq i \leq n$ $A_i = \emptyset$.
- **Type (a) rules:** For every rule $q_0 a \to v q_j \in P$, we add the rule $\#a_l \to \#\hat{v}_{in_j}$ if $q_j \neq q_0$ or the rule $\#a_l \to \#\hat{v}_{here}$ if $q_j = q_0$ to R_0
- **Type (b) rules:** For every rule $q_i a \to v q_j \in P$, and for every symbol $\hat{b} \in \hat{\Sigma}$ we add the rule $\hat{b}a_l \to \hat{b}\hat{v}_{in_j}$ if $q_i \neq q_j$ or the rule $\hat{b}a_l \to \hat{b}\hat{v}_{here}$ if $q_i = q_j$ to R_i
- **Type (c) rules:** For every region R_i and for every pair of symbols $\hat{a} \in \hat{\Sigma}$ and $b \in \Sigma$ add the following rule $\hat{a}b_l \to \hat{a}b_{here}$
- **Type (d) rules:** For every region R_i such that $q_i \in F$, and for every symbol $\hat{a} \in \hat{\Sigma}$ add the following rule $\hat{a}_r \to \breve{a}_{out}$

- **Type (e) rules:** For every region R_i such that $q_i \notin F$, and for every symbol $\hat{a} \in \hat{\Sigma}$ add the following rule $\hat{a}_r \rightarrow \hat{a}_{out}$
- **Type (f) rules:** Add to R the rules $\{\hat{a}_l \rightarrow a_{here} : a \in \Sigma\}$
- **Type (g) rules:** Add to R the rules $\{\breve{a}_l \rightarrow a_{in_0,out}\}$
- **Type (h) rule:** $\#_l \rightarrow \#_{in_0}$

We will explain the rules in the system as follows: Type (a) rules start the transduction of the string from the initial state. Hence, we use the # symbol as a left delimiter of the string to be transduced. The alphabet $\hat{\Sigma}$ is used to mark the symbols that have been transduced during a derivation process. Type (b) rules simulate the transitions in the transducer. Observe that we use the address in_j to change the state in the finite control and the address *here* to simulate the transducer loops. Type (c) rules are used to block the strings that cannot be completely transduced (observe that the IFT can be non complete and it would not finish the derivation process). Type (d) rules are used to output the transduced strings that arrive to a final state. Here, we use the alphabet $\breve{\Sigma}$ to mark the strings that belong to the language generated by the transducer. Type (e) rules are used to output the transduced strings that arrive to a non final state.

The priorities of the rules in regions R_i keep the following order: Type (a) rules > Type (b) rules > Type (c) rules > Type (d) and Type (e) rules.

The rules of the skin region are explained as follows: Type (f) rules are used to restore the string symbols of the transduced string in order to feed-back the transducer with a new input string (hence, it performs the iteration in the transduction). Type (g) rules are used to restore the symbols from those transduced strings that come from a final state (hence, they belong to the language generated by iterating the transducer). In such a case, one copy of the string is sent out the environment while another copy is sent in the region zero in order to feed-back the transducer. Finally, the rule of type (g) is used to send the transduced string into the initial region to iterate a new transduction.

If a string $w \in L(T)$, then $\#w \in L(\Pi)$. We can observe that the transitions from T are simulated by the P system by means of the rules of type (a) and (b). The iteration is carried out at the skin region by applying rules of type (g) or (h) (after restoring the symbols with rules of type (f). If the transduced string arrives to a final state, then rules of type (g) are applied and the string with the left mark # outputs the system.

Example 2. Let us consider the finite transducer defined through the following transition diagram, with a as the starting symbol

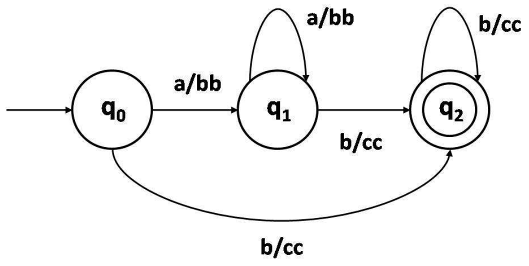
The proposed dogmatic P system is defined with a membrane structure $[[_0]_0, [_1]_1, [_2]_2]$, and the following dogmatic rules

Skin region rules

$r_1 : \hat{a}_l \rightarrow a_{here}$ $r_4 : \breve{a}_l \rightarrow a_{in_0,out}$ $r_{45} : \#l \rightarrow \#_{in_0}$

$r_2 : \hat{b}_l \rightarrow b_{here}$ $r_5 : \breve{b}_l \rightarrow b_{in_0,out}$

$r_3 : \hat{c}_l \rightarrow c_{here}$ $r_6 : \breve{c}_l \rightarrow c_{in_0,out}$



with ρ defined as $\{r_1, r_2, r_3\} > \{r_4, r_5, r_6\} > r_{45}$, and $A = \emptyset$.

Region 0 rules

$r_7 : \#a_l \rightarrow \#\hat{b}\hat{b}_{in_1}$ $r_9 : \hat{a}a_l \rightarrow \hat{a}\hat{b}\hat{b}_{in_1}$ $r_{12} : \hat{a}b_l \rightarrow \hat{a}\hat{c}\hat{c}_{in_2}$
$r_8 : \#b_l \rightarrow \#\hat{c}\hat{c}_{in_2}$ $r_{10} : \hat{b}a_l \rightarrow \hat{b}\hat{b}\hat{b}_{in_1}$ $r_{13} : \hat{b}b_l \rightarrow \hat{b}\hat{c}\hat{c}_{in_2}$
$r_{11} : \hat{c}a_l \rightarrow \hat{c}\hat{b}\hat{b}_{in_1}$ $r_{14} : \hat{c}b_l \rightarrow \hat{c}\hat{c}\hat{c}_{in_2}$

$r_{15} : \hat{a}a_l \rightarrow \hat{a}a_{here}$ $r_{18} : \hat{b}a_l \rightarrow \hat{b}a_{here}$ $r_{21} : \hat{c}a_l \rightarrow \hat{a}a_{here}$
$r_{16} : \hat{a}b_l \rightarrow \hat{a}b_{here}$ $r_{19} : \hat{b}b_l \rightarrow \hat{b}b_{here}$ $r_{22} : \hat{c}b_l \rightarrow \hat{c}b_{here}$
$r_{17} : \hat{a}c_l \rightarrow \hat{a}c_{here}$ $r_{20} : \hat{b}c_l \rightarrow \hat{b}c_{here}$ $r_{23} : \hat{c}c_l \rightarrow \hat{c}c_{here}$

$r_{24} : \hat{a}_r \rightarrow \hat{a}_{out}$
$r_{25} : \hat{b}_r \rightarrow \hat{b}_{out}$
$r_{26} : \hat{c}_r \rightarrow \hat{c}_{out}$

with ρ_0 defined as $\{r_7, r_8\} > \{r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}\} > \{r_{15}, r_{16}, r_{17}, r_{18}, r_{19}, r_{20}, r_{21}, r_{22}, r_{23}\} > \{r_{24}, r_{25}, r_{26}\}$, and $A_0 = \{\#a\}$

Region 1 rules

$r_{27} : \hat{a}a_l \rightarrow \hat{a}\hat{b}\hat{b}_{here}$ $r_{30} : \hat{a}b_l \rightarrow \hat{a}\hat{c}\hat{c}_{in_2}$
$r_{28} : \hat{b}a_l \rightarrow \hat{b}\hat{b}\hat{b}_{here}$ $r_{31} : \hat{b}b_l \rightarrow \hat{b}\hat{c}\hat{c}_{in_2}$
$r_{29} : \hat{c}a_l \rightarrow \hat{c}\hat{b}\hat{b}_{here}$ $r_{32} : \hat{c}b_l \rightarrow \hat{c}\hat{c}\hat{c}_{in_2}$

$r_{33} : \hat{a}a_l \rightarrow \hat{a}a_{here}$ $r_{36} : \hat{b}a_l \rightarrow \hat{b}a_{here}$ $r_{39} : \hat{c}a_l \rightarrow \hat{c}a_{here}$
$r_{34} : \hat{a}b_l \rightarrow \hat{a}b_{here}$ $r_{37} : \hat{b}b_l \rightarrow \hat{b}b_{here}$ $r_{40} : \hat{c}b_l \rightarrow \hat{c}b_{here}$
$r_{35} : \hat{a}c_l \rightarrow \hat{a}c_{here}$ $r_{38} : \hat{b}c_l \rightarrow \hat{b}c_{here}$ $r_{41} : \hat{c}c_l \rightarrow \hat{c}c_{here}$

$r_{42} : \hat{a}_r \rightarrow \hat{a}_{out}$
$r_{43} : \hat{b}_r \rightarrow \hat{b}_{out}$
$r_{44} : \hat{c}_r \rightarrow \hat{c}_{out}$

with ρ_1 defined as $\{r_{27}, r_{28}, r_{29}, r_{30}, r_{31}, r_{32}\} > \{r_{33}, r_{34}, r_{35}, r_{36}, r_{37}, r_{38}, r_{39}, r_{40}, r_{41}\} > \{r_{42}, r_{43}, r_{44}\}$, and $A_1 = \emptyset$

Region 2 rules

$r_{45} : \hat{a}b_l \rightarrow \hat{a}\hat{c}\hat{c}_{here}$
$r_{46} : \hat{b}b_l \rightarrow \hat{b}\hat{c}\hat{c}_{here}$
$r_{47} : \hat{c}b_l \rightarrow \hat{c}\hat{c}\hat{c}_{here}$

$r_{48} : \hat{a}a_l \rightarrow \hat{a}a_{here}$ $r_{51} : \hat{b}a_l \rightarrow \hat{b}a_{here}$ $r_{54} : \hat{c}a_l \rightarrow \hat{c}a_{here}$
$r_{49} : \hat{a}b_l \rightarrow \hat{a}b_{here}$ $r_{52} : \hat{b}b_l \rightarrow \hat{b}b_{here}$ $r_{55} : \hat{c}b_l \rightarrow \hat{c}b_{here}$
$r_{50} : \hat{a}c_l \rightarrow \hat{a}c_{here}$ $r_{53} : \hat{b}c_l \rightarrow \hat{b}c_{here}$ $r_{56} : \hat{c}c_l \rightarrow \hat{c}c_{here}$

$r_{57} : \hat{a}_r \rightarrow \breve{a}_{out}$
$r_{58} : \hat{b}_r \rightarrow \breve{b}_{out}$
$r_{59} : \hat{c}_r \rightarrow \breve{c}_{out}$
with ρ_2 defined as $\{r_{45}, r_{46}, r_{47}\} > \{r_{48}, r_{49}, r_{50}, r_{51}, r_{52}, r_{53}, r_{54}, r_{55}, r_{56}\} > \{r_{57}, r_{58}, r_{59}\}$, and $A_2 = \emptyset$

□

From the previous proposed P system and other works previously referred we get the following result.

Theorem 1. Every recursively enumerable language can be generated by a dogmatic P system.

Proof. The result comes from the simulation of IFTs by dogmatic P systems that we have proposed before. Given that any recursively enumerable can be generated by an IFT with four states [10] then we have the result. □

5 Conclusions and future work

In this paper we have proposed new kinds of rules for P system in which we have been inspired by the Central Dogma of Molecular Biology. The P systems that we have proposed are a suitable framework to generate languages. We think that these kind of rules will help in the construction of systems for biological simulations due to its inspiration from nature.

Our future research will focus on the power of these systems to transduce formal languages with no iteration. Hence, we will study the simulation of rational and recognizable transductions and the simulation of (restricted) *gsms*. In addition, the framework to accept languages of strings or their Parikh mappings (which is the natural framework of P systems) should be explored too. Finally, due to the relation between IFTs and *Computing by carving* we should explore the possibility of applying membrane systems to that paradigm, as a continuation of a previous work [16].

References

1. H. Bordihn, H. Fernau, M. Holzer, V. Manca, C. Martín-Vide. Iterated sequential transducers as language generating devices. *Theoretical Computer Science* 369, pp 67-81. 2006.
2. J. Castellanos, Gh. Păun, A. Rodríguez-Patón. P systems with worm-objects. In Proc. of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00), pages 64-74, A Coruña, Spain, September 2000. IEEE Computer Society.
3. G. Ciobanu, G. Păun, G. Stefănescu. P Transducers. *New Generation Computing* 24, pp 1-28, 2006.
4. W. W. Cohen. A Computer Scientist's Guide to Cell Biology. Springer, 2007.
5. S.R. Krishna, K. Lakshmanan, R. Rama. Hybrid P systems. *Romanian Journal of Information Science and Technology*, 4(1-2):111-123, 2001.
6. S.R. Krishna, R. Rama. Insertion-Deletion P systems. Proc. of Workshop on DNA-Based Computers, pp 360-370, Springer LNCS 2340. 2002.
7. M. Madhu, K. Krithivasan. A note on hybrid P systems. *Grammars*, 5(3):239-244, December 2002.
8. V. Manca. On the Generative Power of Iterated Transduction. In *Words, Semigroups, & Transductions* (M. Ito, G. Păun, S. Yu, eds.) pp 315-327. World Scientific, 2001.
9. V. Manca, C. Martín-Vide, Gh. Păun. New computing paradigms suggested by DNA computing: computing by carving. *BioSystems* 52, pp 47-54. 1999.
10. V. Manca, C. Martín-Vide, Gh. Păun. Iterated GSM Mappings: A Collapsing Hierarchy. In *Jewels are Forever* (J. Karhumäki et al., eds.) pp 182-193. Springer, 1999.
11. C. Martín-Vide, Gh. Păun. String-objects in P systems. In Proc. of Algebraic Systems, Formal Languages and Computations Workshop, pages 161-169, Kyoto, 2000. RIMS Kokyuroku, Kyoto Univ.
12. C. Martín-Vide, Gh. Păun. Language generating by means of membrane systems. *Bulletin of the EATCS*, (75):199-218, October 2001.
13. V. Mitrana, J.M. Sempere. Accepting Evolutionary P Systems. Proc. 10th Workshop on Membrane Computing WMC10. pp 552-555. Universidad de Sevilla. 2009.
14. A. Păun, M. Păun. On Membrane Computing Based on Splicing. *Words, Sequences, Languages*, pp 409-422. Kluwer Academic Publishers. 2000.
15. Gh. Păun. *Membrane Computing. An Introduction.* Springer. 2002.
16. J.M. Sempere. Computing by Carving with P Systems. A First Approach. In 6th Brainstorming Week on Membrane Computing BWMC6. pp 255-260. Fénix Editora, 2008.

Standardized Proofs of PSPACE-completeness of P Systems with Active Membranes

Petr Sosík[1,2], Alfonso Rodríguez-Patón[1], Lucie Ciencialová[2]

[1] Departamento de Inteligencia Artificial, Facultad de Informática
Universidad Politécnica de Madrid, Campus de Monteganced o s/n
Boadilla del Monte, 28660 Madrid, Spain

[2] Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava, 74601 Opava, Czech Republic

Summary. Two proofs have been shown for P systems with active membranes in previously published papers, demonstrating that these P systems can solve in polynomial time exactly the class of problems **PSPACE**. Consequently, these P systems are equivalent (up to a polynomial time reduction) to Second Machine Class models as the alternating Turing machine or the PRAM computer. These proofs were based on a modified definition of uniform families of P systems. Here we demonstrate that the results remain valid also in the case of standard definitions.

1 Introduction

P systems with active membranes are among computationally most powerful models of P systems. It has been shown that this model, in its standard definition, can solve the PSPACE-complete problem QSAT in a polynomial time [8, 1]. Later on, the paper [10] demonstrated that uniform families of P systems with active membranes can solve in polynomial time exactly the class of problems **PSPACE**. Consequently, these P systems satisfy the *Parallel Computation Thesis* [2]:

$$M\text{-PTIME} = M\text{-NPTIME} = \mathbf{PSPACE}, \qquad (1)$$

where M-(N)PTIME is the class of problems solved in polynomial time by a (non-)deterministic machine M. We recall that computers satisfying (1) form the *second machine class*, whose members are the alternating Turing machine, SIMDAG (also known as SIMD PRAM) and other parallel models [2].

However, the papers [8, 10] used a slightly modified version of definition of uniform families of membrane systems. Besides different structure of definitions, the main functional differences between the definition considered standard and presented, e.g., in [5] were these:

1. While both definitions require each P system – a member of a uniform family – to halt, the standard definition requires also the system to produce a distinguished object *yes* or *no* in the last step, telling whether the computation was accepting or not. Our definition, on the contrary, only required the object *yes* in the accepting case.
2. All members of a family must be produced by one and the same Turing machine in the standard definition, while our formulation allowed that different Turing machines might be used for different family members. This possibility, however, was never actually considered and used in our proofs as this would be clearly contra-intuitive to the commonly accepted sense of uniformity.

Therefore, the modification of the proofs presented here focuses on the first mentioned difference and makes the proofs compatible with the standard definition given in the next section.

2 Definitions

A P system with active membranes [7] is a construct

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

where:

(i) $m \geq 1$;
(ii) V is an alphabet;
(iii) H is a finite set of *labels* for membranes;
(iv) μ is a *membrane structure*, consisting of m membranes, labelled (not necessarily in a one-to-one manner) with elements of H; all membranes in μ are supposed to be neutral;
(v) $w_1, \ldots, w_m$ are strings over V, describing the *multisets of objects* placed in the regions of μ;
(vi) R is a finite set of *developmental rules*, of the following forms:
  (a) $[_h a \to v]_h^{\alpha}$,
    for $h \in H, \alpha \in \{+, -, 0\}, a \in V, v \in V^*$
    (object evolution rules, associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part to the application of these rules nor are they modified by them);
  (b) $a[_h\ ]_h^{\alpha_1} \to [_h b]_h^{\alpha_2}$,
    for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in V$
    (communication rules; an object is introduced into the membrane, maybe modified during this process; also, the polarization of the membrane can be modified, but not its label);

(c) $[_h a\,]_h^{\alpha_1} \to [_h\,]_h^{\alpha_2} b$,
for $h \in H, \alpha_1, \alpha_2 \in \{+,-,0\}, a, b \in V$
(communication rules; an object is sent out of the membrane, maybe modified during this process; also, the polarization of the membrane can be modified, but not its label);

(d) $[_h a\,]_h^{\alpha} \to b$,
for $h \in H, \alpha \in \{+,-,0\}, a, b \in V$
(dissolving rules; in reaction with an object, a membrane can be dissolved, leaving all its object in the surrounding region, while the object specified in the rule can be modified);

(e) $[_h a\,]_h^{\alpha_1} \to [_h b\,]_h^{\alpha_2} [_h c\,]_h^{\alpha_3}$,
for $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+,-,0\}, a, b, c \in V$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, maybe of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects; all the other objects are copied into both resulting membranes);

(f) $[_{h_0}[_{h_1}\,]_{h_1}^{+} \cdots [_{h_k}\,]_{h_k}^{+} [_{h_{k+1}}\,]_{h_{k+1}}^{-} \cdots [_{h_n}\,]_{h_n}^{-}]_{h_0}^{\alpha_2}$
$\to [_{h_0}[_{h_1}\,]_{h_1}^{\alpha_3} \cdots [_{h_k}\,]_{h_k}^{\alpha_3}]_{h_0}^{\alpha_5} [_{h_0}[_{h_{k+1}}\,]_{h_{k+1}}^{\alpha_4} \cdots [_{h_n}\,]_{h_n}^{\alpha_4}]_{h_0}^{\alpha_6}$,
for $n > k \geq 1$, $h_i \in H$, $0 \leq i \leq n$, and $\alpha_2, \ldots, \alpha_6 \in \{+,-,0\}$;
(division of non-elementary membranes; this is possible only if a membrane contains two immediately lower membranes of opposite polarization, $+$ and $-$; the membranes of opposite polarizations are separated in the two new membranes, but their polarization can change; all membranes of opposite polarizations are always separated by applying this rule;
if the membrane labelled h_0 contains other membranes than $h_1, \ldots, h_n$ specified above, then they must have neutral charges in order to make this rule applicable; these membranes are duplicated and then become part of the content of both copies of membrane h_0).

All the above rules are applied in parallel, but at one step, an object a can be subject to only one rule of type (a)–(e) and a membrane h can be subject to only one rule of type (b)–(f). In the case of type (f) rules, this means that none of the membranes $h_0, \ldots, h_n$ listed in the rule can be simultaneously subject to another rule of type (b)–(f). However, this restriction do not apply to membranes with neutral charge contained in h_0. In general, an application of the rules is performed as follows:

1. In every step, first the rules are assigned to objects and membranes in a maximal way (any object and membrane which can evolve by a rule of any form, should evolve), and then all the rules are simultaneously applied;
2. all objects and membranes which cannot evolve pass unchanged to the next step;

3. if a rule of type (d), (e) or (f) is applied to a membrane, then rules of type (a) are applied first to its objects and then the resulting objects are further copied/moved in accordance with the (d), (e) or (f) type rule;
4. the skin membrane can neither be dissolved nor divided, nor it can introduce an object from outside (unless stated otherwise). Therefore, we assume that there are only rules of types (a) and (c) associated with the skin membrane.

The membrane structure of Π at a given moment, together with all multisets of objects contained in its regions, form the *configuration* of the system. The $(m+1)$-tuple $(\mu, w_1, \ldots, w_m)$ is the *initial configuration.* We can pass from one configuration to another by using the rules from R according to the principles given above. The computation stops when there is no rule which can be applied to objects and membranes in the last configuration.

In this paper we study the accepting (or recognizer) variant of P systems. A *recognizer P system* solving decision problems must comply with the following requirements: (a) the working alphabet contains two distinguished elements *yes* and *no*; (b) all computations halt; and (c) exactly one of the object *yes* (accepting computation) or *no* (rejecting computation) must be sent to the output region of the system, and only at the last step of each computation. In our case of systems with active membranes, the outer environment of the system is taken as the output region.

2.1 Families of membrane systems

Consider a decision problem $X = (I_X, \theta_X)$ where I_X is a language over a finite alphabet (whose elements are called instances) and θ_X is a total boolean function over I_X.

Definition 1. *[5] A family* $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ *of recognizer membrane systems without input membrane is* polynomially uniform by Turing machines *if there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *from the instance* $w \in I_X$.

In the sequel we will for short denote such a family just as *uniform.*

In this paper we deal with recognizer systems without input membrane, i.e., an instance w of a problem X is encoded into the structure of the P system $\Pi(w)$. The system $\Pi(w)$ is supposed to solve the instance w. Formally, [5] defines the conditions of *soundness* and *completeness* of $\mathbf{\Pi}$ with respect to X. A conjunction of these two conditions ensures that for every $w \in I_X$, if $\theta_X(w) = 1$, then every computation of $\Pi(w)$ is accepting, and if $\theta_X(w) = 0$, then every computation of $\Pi(w)$ is rejecting.

Note that the system $\Pi(w)$ can be generally nondeterministic, i.e, it may have different possible computations, but with the same result. Such a P system is also called *confluent.*

Definition 2. *[5] A decision problem X is solvable in polynomial time by a family of recognizer P systems belonging to a class $\mathcal{R}$ without input membrane $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$, denoted by $X \in \mathbf{PMC}^*_{\mathcal{R}}$, if the following holds:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *The family $\mathbf{\Pi}$ is polynomially bounded; that is, there exists a natural number $k \in \mathbb{N}$ such that for each instance $w \in I_X$, every computation of $\Pi(w)$ performs at most $|w|^k$ steps.*
- *The family $\mathbf{\Pi}$ is sound and complete with respect to X.*

The family $\mathbf{\Pi}$ is said to provide a *semi-uniform solution* to the problem X. In this case, for each instance of X we have a special P system.

3 P Systems with Active Membranes Solving QSAT

The QSAT (satisfiability of quantified propositional formulas) is a well-known **PSPACE**-complete problem. It asks whether or not a given quantified boolean formula in the conjunctive normal form assumes the value *true*. A formula as above is of the form

$$\gamma = Q_1x_1Q_2x_2\dots Q_nx_n(C_1 \wedge C_2 \wedge \dots \wedge C_m), \qquad (2)$$

where each Q_i, $1 \le i \le n$, is either $\forall$ or $\exists$, and each C_j, $1 \le j \le m$, is a *clause* of the form of a disjunction

$$C_j = y_1 \vee y_2 \vee \dots \vee y_r,$$

with each y_k being either a propositional variable, x_s, or its negation, $\sim x_s$. For example, let us consider the propositional formula

$$\beta = Q_1x_1Q_2x_2[(x_1 \vee x_2) \wedge (\sim x_1 \vee \sim x_2)]$$

It is easy to see that it is *true* when $Q_1 = \forall$ and $Q_2 = \exists$, but it is *false* when $Q_1 = \exists$ and $Q_2 = \forall$.

By adding dummy variables, each such formula can be rewritten such that the quantifiers alternate: $Q_1 = \exists$, $Q_2 = \forall$, $Q_3 = \exists$, $Q_4 = \forall$ etc. We assume this normal form for the formulas considered in the sequel.

Theorem 1 ([8]). *There exists a uniform family of recognizer P systems with active membranes providing a semi-uniform solution to QSAT in a time linear in the number of variables and the number of clauses.*

Proof. The following proof differs from the original one published in [8] mostly in omitting the original paragraph 1, modifying paragraph 8 (here paragraph 7) and adding a new paragraph 8.

Consider a propositional formula γ of the form (2) with

$$C_i = y_{i,1} \vee \dots \vee y_{i,p_i},$$

for some $p_i \geq 1$, and $y_{i,j} \in \{x_k, \sim x_k \mid 1 \leq k \leq n\}$, for each $1 \leq i \leq m$, $1 \leq j \leq p_i$. We construct the P system

$$\Pi = (V, H, \mu, w_0, w_1, \ldots, w_m, w_{m+n+1}, R)$$

with the components

$$\begin{aligned} V &= \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{c_i \mid 0 \leq i \leq 4n + 2m + 2\} \cup \{t, s, yes, no\}, \\ H &= \{0, 1, \ldots, m + n + 1\}, \\ \mu &= [_{m+n+1}[_{m+n} \cdots [_1[_0\ ]_0^0]_1^0 \cdots]_{m+n}^0]_{m+n+1}^0, \\ w_0 &= c_0, \\ w_i &= \lambda, \text{ for all } i = 1, 2, \ldots, m + n, \\ w_{m+n+1} &= b, \end{aligned}$$

while the set R contains the following rules:

1. $[_0 c_i \to a_{i/2+1} c_{i+1}]_0^\alpha$, for all $0 \leq i < 2n$, i even, $\alpha \in \{+, -, 0\}$, and

   $[_0 c_i \to c_{i+1}]_0^\alpha$, for all $0 \leq i < 2n$, i odd, or $2n \leq i \leq 2n + m - 1$, $\alpha \in \{+, -, 0\}$

   (we count to $2n + m$, which is the time needed for producing all 2^n truth-assignments for the n variables, as well as 2^n membrane sub-structures which will examine the truth value of formula γ for each of these truth-assignments; this counting is done in the central membrane, irrespective which is its polarity; moreover during first n odd steps, symbols $a_1, \ldots a_n$ are subsequently produced);
2. $[_0 a_i]_0^0 \to [_0 t_i]_0^+ [_0 f_i]_0^-$, for all $1 \leq i \leq n$

   (in membrane 0, when it is "electrically neutral" we subsequently choose each variable x_i, $1 \leq i \leq n$, and both values *true* and *false* are associated with it, in the form of objects t_i, f_i, which are separated in two membranes with the label 0 which differ only by these objects t_i, f_i and by their charge);
3. $[_{i+1}[_i\ ]_i^+ [_i\ ]_i^-]_{i+1}^0 \to [_{i+1}[_i\ ]_i^0]_{i+1}^+ [_{i+1}[_i\ ]_i^0]_{i+1}^-$, for all $0 \leq i \leq m + n - 1$

   (division rules for membranes labeled with $0, 1, \ldots, m + n$; the opposite polarization introduced when dividing a membrane 0 is propagated from lower levels to upper levels of the membrane structure and the membranes are continuously divided until also membrane $m+n$ has been divided; this membrane remains polarized and hence may be never divided again; in the following cycle of the division process, the same holds for the membrane $m+n-1$ and so on, resulting in the structure at Figure 1 after $2n + m$ steps);
4. $[_0 c_{2n+m}]_0^0 \to t$

   (after $2n + m$ steps, each copy of membrane 0 is dissolved and the contents is released into the surrounding membrane, which is labeled with 1);

5. $[_j t_i]_j^0 \to t_i$, if x_i appears in clause C_j, $1 \le i \le n, 1 \le j \le m$, and

   $[_j f_i]_j^0 \to f_i$, if $\sim x_i$ appears in clause C_j, $1 \le i \le n, 1 \le j \le m$

   (a membrane with label j, $1 \le j \le m$, is dissolved if and only if clause C_j is satisfied by the current truth-assignment; if this is the case, then the truth values associated with the variables are released in the surrounding membrane, that associated with the next clause, C_{j+1}, otherwise these truth values remain blocked in membrane j and never used at the next steps by the membranes placed above; note that, as we will see immediately, after $2n+m$ steps we have 2^n membrane sub-structures of the form $[_m [_{m-1} \cdots [_1\ ]_1^0 \cdots]_{m-1}^0]_m^0$ working in parallel; each of them is connected to a leaf of the binary tree membrane structure as in Figure 1);
6. $[_{m+1} t]_{m+1}^\alpha \to [_{m+1}\ ]_{m+1}^\alpha t$, $\alpha \in \{+, -\}$

   (together with the truth-assignments, we also have the object t, which can be passed from a level to the upper one only by dissolving membranes; this object reaches the level $m+1$ if only if all membranes in a sub-structure of the form $[_m [_{m-1} \cdots [_1\ ]_1^0 \cdots]_{m-1}^0]_m^0$ are dissolved, which means that the associated truth-assignment has satisfied all the clauses);
7. $[_i t]_i^\alpha \to [_i\ ]_i^0 s$, $[_i t]_i^0 \to [_i\ ]_i^0 t$, if $Q_{m+n+2-i} = \forall$, $\alpha \in \{+, -\}$, $m+2 \le i \le m+n$, and

   $[_i t]_i^\alpha \to [_i\ ]_i^0 t$, if $Q_{m+n+2-i} = \exists$, $\alpha \in \{+, -\}$, $m+2 \le i \le m+n$

   (a membrane $[_i\ ]_i$ corresponds to the quantifier $Q_j x_j$, where $j = m+n+2-i$; if $Q_j = \forall$, the object t is passed to the upper level only if it comes from both lower level membranes, i.e. the respective clauses are satisfied for both truth values of x_j; if $Q_j = \exists$, then the object t coming from lower level is sent up);
8. $[_{m+n+1} c_i \to c_{i+1}]_{m+n+1}^0$, for all i, $0 \le i < 4n + 2m + 2$,

   $[_{m+n+1} c_{4n+2m+2}]_{m+n+1}^0 \to [_{m+n+1}]_{m+n+1}^- no$,

   $[_{m+n+1} t]_{m+n+1}^0 \to [_{m+n+1}]_{m+n+1}^+ yes$

   (objects c_i in the region enclosed by the skin membrane act as a clock; if the object t reaches this region within $4n + 2m + 2$ steps, signalling that the formula evaluates to *true*, then the object *yes* is expeled from the system, otherwise the object *no* is expeled after $4n + 2m + 2$ steps. In both cases the systems immediately halts.

From the previous explanations one can see that the object *yes* (*no*) leaves the system in the last step if and only if formula γ evaluates to *true* (*false*, respectively). This is achieved in $3n + \lfloor n/2 \rfloor + 2m + 2$ steps:

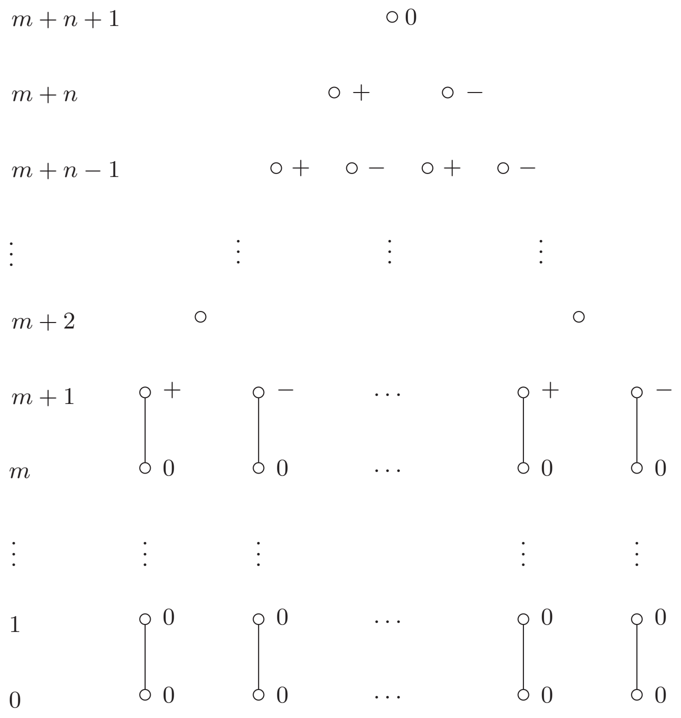


Fig. 1. The membrane structure of the system Π after $2n+m$ steps.

- in $2n+m$ steps we create the membrane structure at Figure 1 (as well as the 2^n different truth-assignments)
- then we dissolve all membranes 0 (one step)
- we check the satisfiability of each clause for each truth-assignment, in parallel in the 2^n sub-structures ($m+1$ steps)
- we check whether all quantifiers are satisfied by propagating objects t through the indicated binary tree structure; one step is need for each of $\lceil n/2 \rceil$ quantifiers $\exists$, while two steps are necessary for each of $\lfloor n/2 \rfloor$ quantifiers $\forall$.

The arguments given above ensure that the system Π is polynomially bounded and that the family of these P systems is complete and sound with respect to the problem QSAT. Finally, the family is polynomially uniform by Turing machines as the above construction can be performed by an algorithm which would run on a classical computer (and hence also on Turing machine) in a polynomial time, having as input a propositional formula γ (an instance of QSAT) and which would output the description of the system Π. Note that both the size of the alphabet V and the number of membranes in the initial configuration is $\mathcal{O}(n+m)$, and the number of rules is $\mathcal{O}(nm)$, which determines the time necessary for the construction. Since the construction can be done step-by step without a need to store previous steps, the space needed is $\mathcal{O}(\log n + \log m)$.

Corollary 1 ([8]). PSPACE $\subseteq$ PMC$^{S}_{\mathcal{AM}}$.

4 Simulation of P Systems with Active Membranes in Polynomial Space

In this section we show that the inclusion reverse to Corollary 1 hold as well. We employ the technique of reverse-time simulation. Instead of simulating a computation of a P system from its initial configuration onwards (which would require an exponential space for storing configurations), we create a recursive function which returns the state of any membrane h after a given number of steps. The recursive calls evaluate contents of the membranes interacting with h in a reverse time order (towards the initial configuration). In such a manner we do not need to store a state of any membrane, but instead we calculate it recursively whenever it is needed. In this way a result of any $T(n)$-time-bounded computation of a recognizer P system with active membranes can be found in a space polynomial to $T(n)$.

Theorem 2 ([10]). **PMC$^{S}_{\mathcal{AM}}$ $\subseteq$ PSPACE**.

Proof. The proof of this result published in [10] remains unchanged under the standard definition, except the paragraph at p. 149 under the subtitle “Space complexity of the simulation”, starting with ”Consider an instance of a size s. . . ”. This paragraph should be reformulated as follows:

Consider a decision problem which is, by assumption, solved by a uniform family of P system with active membranes in a semi-uniform way. Each instance of a size s is solved by a P system $\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$ of size $s^{\mathcal{O}(1)}$, a member of the family. The result of computation of Π can be calculated with the aid of the function `State`. Let h_0 be the skin membrane of Π. One can subsequently calculate `State`(h_0, n) for $n = 0, 1, 2 \ldots$ until the object *yes* or *no* is expelled from h_0 using the rule of type (c). We determine the space complexity of the function `State`. Let. . . □

Together with Corollary 1 we obtain the parallel computation thesis for uniform families of recognizer P systems with active membranes:

Corollary 2. **PMC$^{S}_{\mathcal{AM}}$ = PSPACE**.

5 Concluding Remarks

Since the publication of papers [8, 10], similar results linking the class PSPACE with other types of membrane systems have been presented, see, e.g., [3, 9]. The proof technique we have used in Theorem 2 is applicable also to other variants of P systems.

Finally, we note that the characterization of power of non-confluent P systems with active membranes remains still open. The presented proof cannot be simply adapted to this case by using a non-deterministic Turing machine. The reason is that we cannot store non-deterministic choices of such a P system along a chosen trace of computation, as this would require an exponential space. It is possible that non-confluent P systems with active membranes might capture in polynomial time the class **NEXPTIME**.

Acknowledgements

Authors are grateful to Mario Pérez-Jiménez for discussions and suggestions concerning complexity issues and uniform families of membrane systems. The research was partially supported by the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2009–14421, by the program I3, by the Comunidad de Madrid (grant No. CCG06-UPM/TIC-0386 to the LIA research group), by the Czech Science Foundation, grant No. 201/09/P075, and by the Silesian University in Opava, grant No. SGS/4/2010.

References

1. A. Alhazov, C. Martín-Vide, and L. Pan. Solving a PSPACE-complete problem by P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.
2. P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 1–66. Elsevier, Amsterdam, 1990.
3. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, and X. Zhang. Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. In Martínez-del-Amor et al. [4], pages 1–27. Volume 2.
4. M.A. Martínez-del-Amor, E.F. Orejuela-Pinedo, G. Păun, I. Pérez-Hurtado, and A. Riscos-Núnez, editors. *Seventh Brainstorming Week on Membrane Computing*, Sevilla, 2009. Fenix Editora.
5. M.J. Pérez-Jiménez. A computational complexity theory in membrane computing. In Păun et al. [6], pages 125–148.
6. G. Păun, M.J. Pérez-Jiménez, A. Riscos-Núnez, G. Rozenberg, and A. Salomaa, editors. *Membrane Computing, 10th International Workshop, WMC 2009*, volume 5957 of *Lecture Notes in Computer Science*, Berlin, 2010. Springer.
7. Gh. Păun. P systems with active membranes: attacking NP-complete problems. *J. Automata, Languages and Combinatorics*, 6 (1):75–90, 2001.
8. P. Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, 2003.
9. P. Sosík, A. Păun, A. Rodríguez-Patón, and D. Pérez. On the power of computing with proteins on membranes. In Păun et al. [6], pages 448–460.
10. P. Sosík and A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *J. Comput. System Sci.*, 73(1):137–152, 2007.

When Matrices Meet Brains

Xiangxiang Zeng[1,3], Henry Adorna[2],
Miguel Ángel Martínez-del-Amor[3] Linqiang Pan[1]

[1] Key Laboratory on Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074 Hubei, Peoples Republic of China
[2] Department of Computer Science (Algorithms and Complexity)
University of the Philippines
Diliman 1101 Quezon City, Philippines
[3] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

Summary. Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. In this work, a discrete structure representation of SN P systems is proposed. Specifically, matrices are used to represent SN P systems. In order to represent the computations of SN P systems by matrices, configuration vectors are defined to monitor the number of spikes in each neuron at any given configuration; transition net gain vectors are also introduced to quantify the total amount of spikes consumed and produced after the chosen rules are applied. Nondeterminism of the systems is assured by a set of spiking transition vectors that could be used at any given time during the computation. With such matrix representation, it is quite convenient to determine the next configuration from a given configuration, since it involves only multiplying vectors to a matrix and adding vectors.

1 Introduction

Membrane computing was initiated by Păun [6] and has developed very rapidly (already in 2003, ISI considered membrane computing as "fast emerging research area in computer science", see `http://esi-topics.com`). It aims to abstract computing ideas (data structures, operations with data, computing models, etc.) from the structure and the functioning of single cell and from complexes of cells, such as tissues and organs including the brain. The obtained models are distributed and parallel computing devices, called *P systems*. For updated information about membrane computing, please refer to [8].

This work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short) [3]. SN P systems were inspired by the neurophysiological behavior of neurons (in brain) sending electrical impulses along axons to other neurons, with the aim of incorporating specific ideas from spiking neurons into membrane computing. Generally speaking, in an SN P system the processing elements are called *neurons* and are placed in the nodes of a directed graph, called the *synapse graph*. The content of each neuron consists of a number of copies of a single object type, namely the *spike*. Each neuron may also contain rules which allow to remove a given number of spikes from it, or send spikes (possibly with a delay) to other neurons. The application of every rule is determined by checking the content of the neuron against a regular set associated with the rule.

Representation of P systems by discrete structures has been one topic in the field of membrane computing. One of the promising discrete structures to represent P systems is matrix. Models with matrices as their representation have been helpful to physical scientists – biologist, chemists, physicists, engineers, statisticians, and economists – solving real world problems. Recently, matrix representation was introduced to represent a restricted form of cell-like P systems without dissolution (where only non-cooperate rules are used) [2]. It was proved that with algebraic representation P systems can be easily simulated and computed backward (that is, to find all the configurations that produce a given one in one computational step).

In this work, a matrix representation of SN P systems without delay is proposed, where configuration vectors are defined to represent the number of spikes in neurons; spiking vectors are used to denote which rules will be applied; a spiking transition matrix is used to describe the skeleton of system; transition net gain vectors are also introduced to quantify the total amount of spikes consumed and produced after the chosen rules are applied. With these algebraic representation, matrix transition can be used to compute the next configuration from a given configuration.

The rest of this paper is organized as follows. In the next section, we introduce the definition of SN P systems. Section 3 presents the matrix representation of SN P systems. Section 4 illustrates how to represent the computation of SN P system by matrix transition. Conclusions and remarks are given in Section 5.

2 Spiking Neural P Systems

In this section, a restricted variant of SN P systems, SN P systems without delay, is introduced.

Definition 1. *An SN P system without delay, of degree $m \geq 1$, is a construct of the form*

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

where:

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

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:
a) $n_i \geq 0$ is the initial number of spikes contained in σ_i;
b) R_i is a finite set of rules of the following two forms:
(1) $E/a^c \rightarrow a^p$, where E is a regular expression over a, and $c \geq 1$, $p \geq 1$, with the restriction $c \geq p$;
(2) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p$ of type (1) from R_i, $a^s \notin L(E)$;
3. $syn = \{(i,j) \mid 1 \leq i,j \leq m,\ i \neq j\}$ (synapses between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the input and output neurons, respectively.

The rules of type (1) are spiking (or called firing) rules, which are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a^p \in R_i$ can be applied. This means that consuming (removing) c spikes (thus only $k - c$ spikes remain in σ_i), the neuron is fired, and it produces p spikes; these spikes are transported to all neighbor neurons by outgoing synapses.

The rules of type (2) are forgetting rules; they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \rightarrow \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \rightarrow a$ has $E = a^c$, then it is written in the simplified form $a^c \rightarrow a$.

In each time unit, if a neuron σ_i can apply one of its rules, then a rule from R_i must be applied. Since two spiking rules, $E_1/a^{c_1} \rightarrow a^{p_1}$ and $E_2/a^{c_2} \rightarrow a^{p_2}$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules are applicable in a neuron, and in that case, only one of them is chosen and applied non-deterministically. However, note that, by definition, if a spiking rule is applicable, then no forgetting rule is applicable, and vice versa.

Thus, the rules are applied in the sequential manner in each neuron, at most one in each step, but neurons function in parallel with each other. It is important to notice that the applicability of a rule is established based on the total number of spikes contained in the neuron.

A configuration of the system is described by the number of spikes present in each neuron. Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be applied. The result of a computation is the number of steps elapsed between the first two spikes sent by the output neuron during the computation.

3 Matrix Representation of SN P System

In this section, a matrix representation of SN P system is given.

As mentioned in the above section, a configuration of the system is described by the number of spikes present in each neuron. Here, vectors are used to represent configurations.

Definition 2 (Configuration Vectors). *Let Π be an SN P system with m neurons, the vector $C_0 = (n_1, n_2, \ldots, n_m)$ is called the* **initial configuration vector** *of Π, where n_i is the amount of the initial spikes present in neuron σ_i, $i = 1, 2, \ldots, m$ before the computation starts.*

For any $k \in \mathbb{N}$, the vector $C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)})$ is called the ***k*th configuration vector** *of the system, where $n_i^{(k)}$ is the amount of spikes in neuron σ_i, $i = 1, 2, \ldots, m$ after the kth step in the computation.*

In order to describe which rules are chosen and applied in each configuration, *spiking vector* is defined.

Definition 3 (Spiking Vectors). *Let Π be an SN P system with m neurons and n rules, $m \leq n < \infty$. Assume a total order $d : 1, \ldots, n$ is given for all the n rules, so the rules can be referred as $r_1, \ldots, r_n$. A* **spiking vector** *$s^{(k)}$ is defined as follows:*

$$s^{(k)} = (r_1^{(k)}, r_2^{(k)}, \ldots, r_n^{(k)}),$$

where:

$$r_i^{(k)} = \begin{cases} 1, \text{ if the regular expression } E_i \text{ of the rule } r_i \text{ is satisfied} \\ \quad \text{and the rule } r_i \text{ is chosen and applied;} \\ 0, \text{ otherwise.} \end{cases}$$

In particular, $s^{(0)} = (r_1^{(0)}, r_2^{(0)}, \ldots, r_n^{(0)})$ is called the **initial spiking vector**.

In each configuration, when the chosen rules are applied, the change of the number of spikes in each neuron is represented by *spiking transition matrix.*

Definition 4 (Spiking Transition Matrix). *Let Π be an SN P system with m neurons and n rules, $d : 1, \ldots, n$ a total order given for all the n rules. The* **spiking transition matrix** *of the system Π, M_Π, is defined as follows:*

$$M_\Pi = [a_{ij}]_{n \times m},$$

where:

$$a_{ij} = \begin{cases} -c, \text{ if rule } r_i \text{ is in neuron } \sigma_j \text{ and it is applied consuming } c \text{ spikes;} \\ \ \ p, \text{ if rule } r_i \text{ is in neuron } \sigma_s \ (s \neq j \text{ and } (s,j) \in syn) \\ \quad \text{and it is applied producing } p \text{ spikes;} \\ \ \ 0, \text{ if rule } r_i \text{ is in neuron } \sigma_s \ (s \neq j \text{ and } (s,j) \notin syn). \end{cases}$$

In a spiking transition matrix, the row i is associated with the rule $r_i : E/a^c \to a^p$. Assume that the rule r_i is in neuron σ_j. When the rule r_i is applied, it consumes c spikes in neuron σ_j; neuron σ_s ($s \neq j$ and $(j, s) \in syn$) receives p spikes from neuron σ_j; neuron σ_s ($s \neq j$ and $(j, s) \notin syn$) receives no spike from neuron σ_j. By the definition of spiking transition matrix, the entry in the position (i, j) is a negative number; the other entries in the row i are non-negative numbers. So the following observation holds:

Observation 1 *Each row of a spiking transition matrix has exactly one negative entry.*

In a spiking transition matrix, the column i is associated with neuron σ_i. For an SN P system, without loss of generality, it can be assumed that each neuron has at least one rule inside (if a neuron has no rule inside, it just stores spikes, sending no spikes to other neurons or environment, so it can be deleted without any influence to the computational result of the system). Assume the rules to be $r_m, r_n, \ldots$. The rules consume spikes in neuron σ_i when they are applied. So the corresponding entries $(m, i), (n, i), \ldots$ in the spiking transition matrix are negative numbers, and the following observation holds:

Observation 2 *Each column of a spiking transition matrix has at least one negative entry.*

4 Computing via Matrices

In this section, it is shown how the computation of SN P system can be represented by operations on matrices, by using the matrix representation defined in the above section.

First, we provide an example before we define formally the matrix operations for an SN P system.

Example 1. Let us consider an SN P system $\Pi = (\{a\}, \sigma_1, \sigma_1, \sigma_3, syn, out)$ that generates the set $\mathbb{N}$ of natural numbers excluding 1, where $\sigma_1 = (2, R_1)$, with $R_1 = \{a^2/a \to a, a^2 \to a\}$; $\sigma_2 = (1, R_2)$, with $R_2 = \{a \to a\}$; $\sigma_3 = (1, R_3)$, with $R_3 = \{a \to a, a^2 \to \lambda\}$; $syn = \{(1, 2), (1, 3), (2, 1), (2, 3)\}$; $out = 3$. Π is also represented graphically in Figure 1, which may be easier to understand .

In order to represent the above SN P system Π in a matrix, firstly, we set a total order for all the rules in the system, which can be seen in Figure 1. With this order, the rules can be denoted by $r_1, \ldots, r_5$.

Let $M_{\Pi_1} = [a_{ij}]_{5\times 3}$ be the spiking transition matrix for Π. As defined in Section 3, row i of M_Π is associated with rule $r_i : E/a^c \to a^p, c \geq 1, p \geq 0$ in system Π. The entries a_{i1}, a_{i2}, a_{i3} are the amount of spikes which neurons $\sigma_1, \sigma_2, \sigma_3$ will get (or consume) when rule r_i is applied.

We have the following spiking transition matrix for the SN P system Π in Figure 1.

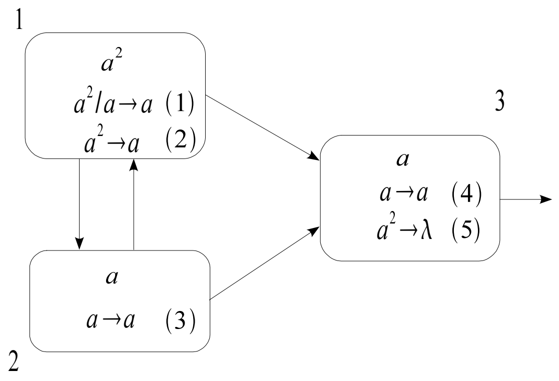


Fig. 1. An SN P system Π that generates the set $\mathbb{N} - \{1\}$

$$M_\Pi = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} \qquad (1)$$

Initially, neurons σ_1, σ_2, σ_3 have 2, 1, 1 spikes, respectively. According to Definition 2, the initial configuration vector for system Π would be $C_0 = (2, 1, 1)$. We also get the initial spiking vector by Definition 3: since neuron σ_1 has two rules r_1 and r_2 that could possibly be applied in the initial transition, one of the rules could be chosen. The initial spiking transition vector would be $(1, 0, 1, 1, 0)$ or $(0, 1, 1, 1, 0)$. Note here that we cannot use rule r_5 because the regular expression a^2 is not satisfied in neuron σ_3.

If rule $r_1 : a^2/a \to a$ is applied, it consumes one spike in neuron σ_1 and sends 1 spike to neurons σ_2 and σ_3, respectively; at the same time, neuron σ_2 sends 1 spike to neurons σ_1 and σ_3. In this step, the net gain of neuron σ_1 is 0 spike (it consumes 1 spike by r_1 and receives 1 spike from neuron σ_2); the net gain of neuron σ_2 is 0 spike (it consumes 1 spike by r_3 and receives 1 spike from neuron σ_1); the net gain of neuron σ_3 is 1 spike (it consumes 1 spike by rule r_5 and receives 2 spikes from neurons σ_1 and σ_2, respectively). After this step, the numbers of spikes in neurons σ_1, σ_2, σ_3 are 2, 1, 2, respectively.

Our illustration above served as witness to the following results.

Definition 5 (Transition Net Gain Vector). *Let Π be an SN P system with m neurons and n rules, $m \leq n < \infty$, $C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)})$ the kth configuration vector of an SN P system Π. We define*

$$NG^{(k)} = C_{k+1} - C_k,$$

as the **transition net gain vector** *at step k.*

Lemma 1. *Let Π be an SN P system with m neurons and n rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the n rules, M_Π the spiking transition matrix of Π, $s^{(k)}$ the spiking vector at step k. Then the transition net gain vector at step k can be obtained by*

$$NG^{(k)} = s^{(k)} \cdot M_\Pi. \tag{2}$$

Proof. Equation (2) implies that $NG^{(k)} = (g_1, g_2, \ldots, g_m)$, such that $g_j = \Sigma_{i=1}^{n} r_i^{(k)} a_{ij}$, for all $j = 1, 2, \ldots, m$. Note that the spiking vector $s^{(k)}$ is a $\{0, 1\}$-vector that identifies the rules in each neuron that would be applied at step k. Thus, g_j represents the total amount of spikes obtained and consumed by neuron σ_j after applying the rules identified by $s^{(k)}$. Therefore, we have $(n_1^{(k+1)}, n_2^{(k+1)}, \ldots, n_m^{(k+1)}) = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)}) + (g_1, g_2, \ldots, g_m)$, that is, $C_{k+1} = C_k + NG^{(k)}$.

Theorem 1. *Let Π be an SN P system with m neurons and n rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the n rules, M_Π the spiking transition matrix of Π, $C^{(k)}$ the kth configuration vector, $s^{(k)}$ the spiking vector at step k, then every configuration C_k of Π can be obtained by*

$$C_k = C_{k-1} \; + \; s^{(k-1)} \cdot M_{SNP}. \tag{3}$$

Proof. This results follows directly from the preceding Lemma.

Let us go back to the example show in Figure 1. Given the initial configuration vector $C_0 = (2, 1, 1)$, the next configuration of system Π can be computed as follows:

If we choose the rules r_1, r_3, r_4 to apply, the spiking vector will be $s^{(0)} = (1, 0, 1, 1, 0)$, then the next configuration can be obtained by:

$$C_1 = (2 \quad 1 \quad 1) \; + \; (1 \quad 0 \quad 1 \quad 1 \quad 0) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (2 \quad 1 \quad 2) \tag{4}$$

In the next step, we choose r_1, r_3, r_5 to apply, the spiking vector is $(1, 0, 1, 0, 1)$, then the next configuration is:

$$C_2 = (2 \quad 1 \quad 2) \; + \; (1 \quad 0 \quad 1 \quad 0 \quad 1) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (2 \quad 1 \quad 2) \tag{5}$$

So, if we choose the spiking vector to be $(1, 0, 1, 0, 1)$, the transition net gain vector will be:

$$NG = (1 \quad 0 \quad 1 \quad 0 \quad 1) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (0 \quad 0 \quad 0) \tag{6}$$

Equation (6) means that the configuration of the system will remain unchanged as long as we choose rules r_1, r_3 and r_5 to apply. However, at any moment, starting from the first step of computation, neuron σ_1 can choose to use the rule $r_2 : a^2 \to a$, in this case system will go to another configuration, we do not want to check the computation in detail here but leave this task to the readers.

Finally, the following Corollary is a direct consequence of the preceding Theorem.

Corollary 1. *Let Π be an SN P system with m neurons and n rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the n rules, M_Π the spiking transition matrix of Π, $C^{(k)}$ the kth configuration vector, $s^{(k)}$ the spiking vector at step k, the previous configuration C_{k-1} can be obtained by*

$$C_{k-1} = C_k \ - \ s^{(k-1)} \cdot M_\Pi. \tag{7}$$

In our matrix representation, we do not include the environment. This idea can be incorporated in the definition of a spiking transition matrix by introducing a so-called **augmented spiking transition matrix.**

Definition 6 (Augmented Transition Spiking Matrix). *Let Π be an SN P system with m neurons and n rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the n rules, M_Π the $n \times m$ spiking transition matrix of Π. We define an* **augmented spiking transition matrix** *as follows:*

$$[M_\Pi \mid e]_{n\times(m+1)},$$

where $e = (e_1, e_2, \ldots, e_n)^T$ is a column representing environment, where:

$$e_i = \begin{cases} p, & \text{if rule } r_i \text{ is in the output neuron and it is applied producing } p \text{ spikes;} \\ 0, & \text{if rule } r_i \text{ is not in the output neuron.} \end{cases}$$

Correspondingly, we define

$$C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)}, n_e^{(k)}),$$

to be **augmented configuration vector** after the kth step in the computation, where $n_i^{(k)}$ is the amount of spikes in neuron σ_i, for all $i = 1, 2, \ldots, m$, $n_e^{(k)}$ is the amount of spikes collected by environment. Using this vector instead of the configuration vector defined in Section 3 simply allows us to monitor the output of a system.

5 Conclusions and Remarks

In this paper, we have found a universal algebraic representation for SN P systems: for every SN P system without delay, configuration vectors are defined to represent the number of spikes in neurons; spiking vectors are used to denote which rules will be applied; a spiking transition matrix is used to describe the skeleton of system. Such algebraic representation turns out to be a reasonable representation of SN P systems. It is shown that matrix computation is convenient for deciding the next configuration of our systems.

It is not difficult to see that such matrix representation is also suitable for other variants of SN P systems, such as asynchronous SN P systems [1], SN P systems with exhaustive use of rules [4], and so on. The spiking transition matrix is related to the structure of system only, so that the elements of the matrix are determined initially. During the computation of a system, it is only necessary to decide the spiking vector by checking the current configuration vector and the regular expressions of rules. Thus, it is easy to program for computer application, which can offer as a powerful tool for the simulation and analysis of these systems.

Anyway, there are many issues still worth investigating. Another research line would be of interest to see whether matrix can be used for biological neural networks, for example, human brain. A problem is how to capture the feature that neurons have refractory time (in the present paper, we have not considered this feature in our model).

Acknowledgements

The work of X. Zeng and L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60533010), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), and HUST-SRF (2007Z015A).

References

1. M. Cavaliere, O. Egecioglu, O. H. Ibarra, S. Woodworth, M. Ionescu, and Gh. Păun, Asynchronous spiking neural P systems, *Theoretical Computer Science*, 2009, 410, 2352–2364.
2. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, Computing backwards with P systems, *WMC10*, Curtea de Arges, Romania, 2009, 282–295.
3. M. Ionescu, Gh. Păun, and T. Yokomori, Spiking neural P systems. *Fundamenta Informaticae*, 2006, 71(2–3), 279–308.
4. M. Ionescu, Gh. Păun, and T. Yokomori, Spiking neural P systems with exhaustive use of rules, *International Journal of Unconventional Computing*, 2007, 3, 135–154.
5. J. K. Nelson and J. C. McCormac, Structural Analysis: Using Classical and Matrix Methods, 3rd Edition, Wiley, 2003.

6. Gh. Păun, Computing with membranes, *Journal of Computer and System Science*, 2000, 61(1), 108–143.
7. Gh. Păun, Membrane Computing – An Introduction, Springer-Verlag, Berlin, 2002.
8. The P System Web Page: http://ppage.psystems.eu.

An Approximate Algorithm Combining P Systems and Ant Colony Optimization for Traveling Salesman Problems

Ge-xiang Zhang[1], Ji-xiang Cheng[2], Marian Gheorghe[3]

[1] School of Electrical Engineering
Southwest Jiaotong University
Chengdu, Sichuan, 610031, P.R.China
`zhgxdylan@126.com`

[2] School of Information Science & Technology
Southwest Jiaotong University
`chengjixiang0106@126.com`

[3] Department of Computer Science
University of Sheffield
Sheffield S1 4DP, UK
`m.gheorghe@dcs.shef.ac.uk`

Summary. This paper proposes an approximate optimization algorithm combining P systems with ant colony optimization, called ACOPS, to solve traveling salesman problems, which are well-known and extensively studied NP-complete combinatorial optimization problems. ACOPS uses the pheromone model and pheromone update rules defined by ant colony optimization algorithms, and the hierarchical membrane structure and transformation/communication rules of P systems. First, the parameter setting of the ACOPS is discussed. Second, extensive experiments and statistical analysis are investigated. It is shown that the ACOPS is superior to Nishida's algorithms and its counterpart ant colony optimization algorithms, in terms of the quality of solutions and the number of function evaluations.

1 Introduction

As a young and vigorous branch of natural computing, membrane computing focuses on abstracting computing models and membrane systems from the structure and the functioning of the living cell as well as from the cooperation of cells in tissues, organs, and other populations of cells [1, 2]. The molecular interactions of neurons inspired the development of the neural P systems [3]. In recent years, the interaction between membrane computing and other nature-inspired computing paradigms has been considered from various perspectives. The integration of meta-heuristic search methodologies and P systems has given birth to membrane algorithms [4], which prove to be efficient and effective ways to solve various real world problems. Generally, there are two main methods utilized in providing solutions to optimization problems, approximate and complete approaches. The complete algorithms are guaranteed to find an optimal solution in bounded time for every finite size instance of an optimization problem and they may require exponential computing time in the worst case for an NP-hard problem [5]. These approaches are ineffective in practical circumstances and other ways of tackling these problems are considered. The approximate algorithms instead, focus on producing good solutions in significantly less time rather than obtaining optimal solutions which are hard to compute. These approaches are very important in solving various continuous and combinatorial optimization problems [5].

The membrane algorithm, or the approximate optimization algorithm based on P systems, is a fertile research direction which explores the great potential of membrane computing as a distributed processing mechanism. Until now, relatively limited work has been produced in this field. In [6, 7, 8], an approximate algorithm using a nested membrane structure (NMS) and a local search technique to solve traveling salesman problems was presented. The algorithm was also applied to obtain good approximate solutions to the min storage problem [9]. In [10, 11], an optimization algorithm combining the NMS and conventional genetic algorithms was presented to solve single-objective and multi-objective numerical optimization problems. In [12], the similarities between distributed evolutionary algorithms and P systems were analyzed and new variants of distributed evolutionary algorithms are suggested and applied for some continuous optimization problems. In our previous work [4, 13], a quantum-inspired evolutionary algorithm based on P systems (QEPS) was proposed to solve knapsack and satisfiability problems. In the QEPS, a one-level membrane structure (OLMS) was introduced and a comparison between the OLMS and the NMS was experimentally investigated. Further variants of the QEPS were applied to analyze radar emitter signals and design digital filters [14, 15, 16]. In [17], the application of membrane algorithms to controller design was discussed. All these investigations support the claim made by Păun and Pérez-Jiménez [18] that the membrane algorithm is a rather new research direction with a well-defined scope, a set of open questions, and therefore further studies are necessary to prove the usefulness of P systems-based approaches for real-world applications.

The already established way of conceiving membrane algorithms is to explore the interactions between P systems and various meta-heuristic techniques for solving different problems; their performance is assessed by comparing the results produced by them and their complexity aspects with those obtained by using already available optimization techniques. In this paper, an approximate optimization algorithm integrating P systems and ant colony optimization techniques, called ACOPS, is proposed in order to solve traveling salesman problems (TSPs). This is the first attempt to investigate the interaction between P systems and swarm intelligence approaches in defining membrane algorithms. We use the pheromone model and pheromone update rules of ant colony optimization (ACO) algorithms, and the hierarchical membrane structure and transformation/communication rules of P systems, to specify the ACOPS algorithm. Also we discuss the parameter setting of the ACOPS. A TSP is a well-known and extensively studied NP-complete combinatorial optimization problem. Experiments conducted on fairly large TSP instances and statistical analysis undertaken show that the ACOPS outperforms Nishida's algorithms and its counterpart ACO algorithms. This work is an example of a successful use of membrane computing, in combination with efficient searching methods, for designing approximate optimization algorithms.

This paper is organized as follows. Section 2 first gives a brief introduction of TSP, P systems, and ACO, and then discusses the ACOPS in detail. Section 3 addresses some discussions with regard to parameter setting and experimental results. Conclusions are drawn in Section 4.

2 ACOPS

This section starts with a brief description of the TSP problem, some basic P systems concepts, and a presentation of the ant optimization algorithm. The rest of this section is dedicated to a presentation of the ACOPS.

2.1 Traveling Salesman Problems

TSP is one of the well-known and most intensively studied combinatorial optimization problems in the areas of optimization, operational research, theoretical computer science, and computational mathematics [19, 20]. A TSP can be described as follows. Given a set C of N cities, i.e., $C = \{c_1, c_2, \cdots, c_N\}$, and a set D of the pairwise travel costs d_{ij}, $i, j = 1, 2, \cdots, N, i \neq j$, i.e., $D = \{d_{ij}\}$, it is requested to find the minimal cost of the path taken by a salesman visiting each of the cities just once and returning to the starting point. More generally, the task is to find a Hamiltonian tour with a minimal length in a connected, directed graph with a positive weight associated to each edge. If $d_{ij} = d_{ji}$, the TSP is symmetric in the sense that traveling from city X to city Y costs just as much as traveling in the opposite direction, otherwise, it is asymmetric.

In the TSP, the cost could be associated with distance, time, money, energy, etc.. A number of industrial problems such as network structure design, machine scheduling, cellular manufacturing, and frequency assignment, can be formulated as TSPs; consequently TSP is often used as a standard benchmark for optimization algorithms [21]. In the theory of computational complexity, TSP belongs to the class of NP-complete problems. In this paper, we will consider the symmetric TSPs, in which the distance is used as the cost.

2.2 P Systems

Various P systems in the literature can be classified in three groups: cell-like P systems, tissue- like P systems and neural-like P systems [22]. A cell-like P system, considered in this paper, is characterized by three components: the membrane structure delimiting compartments, the multisets of abstract objects placed in compartments, and the evolution rules applied to objects. The membrane structure of a cell-like P system, shown in Fig. 1, is a hierarchical arrangement of membranes [1]. The outermost membrane is the skin membrane separating the system from its environment. As usual, there are several membranes inside the skin membrane. Each of these membranes defines a region, which forms a different compartment of the membrane structure and contains a multiset of objects, other membranes and a set of evolution rules. The membrane without any membrane inside is called an elementary membrane.

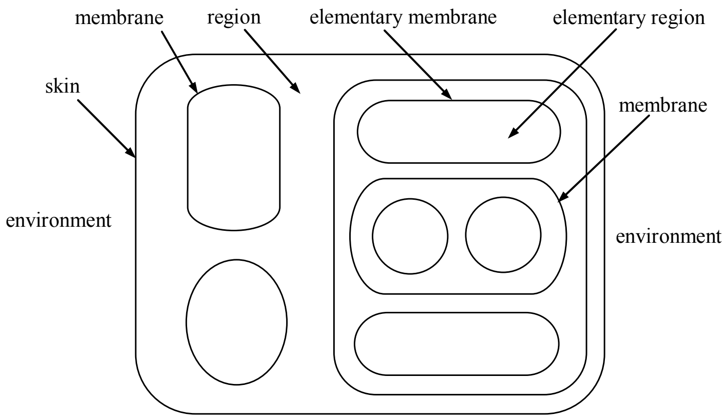


Fig. 1. The membrane structure of a cell-like P system [1]

A cell-like P system with an output set of objects and using transformation and communication rules is formally defined as follows [1, 23]

$$\Pi = (V, T, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, i_0),$$

where

(i) V is an alphabet; its elements are called objects;
(ii) $T \subseteq V$ the output alphabet);
(iii) μ is a membrane structure consisting of m membranes, with the membranes and the regions labelled in a one-to-one manner with elements of a given set Λ – usually the set $\{1, 2, \cdots, m\}$; m is called the degree of Π;
(iv) w_i, $1 \leq i \leq m$, are strings representing multisets over V associated with the regions, $1, 2, \cdots, m$, of μ;
(v) R_i, $1 \leq i \leq m$, are sets of rules associated with the regions, $1, 2, \cdots, m$, of μ;
(vi) i_0 is a number between 1 and m which specifies the output membrane of Pi.

The rules of R_i, $1 \leq i \leq m$, have the form $a \rightarrow v$, where $a \in V$ and $v \in (V \times \{here, out, in\})^*$. The multiset v consists of pairs (b, t), $\in V$ and

$t \in \{here, out, in\}$, where *here* means that b will stay in the region where the rule is used; *out* shows that b exits the region and *in* means that b will be communicated to one of the membranes contained in the current region which is chosen in a non-deterministic way.

A P system, regarded as a model of computation, provides a suitable framework for distributed parallel computation that develops in steps. In the process of computation, multisets of simple objects or are rewritten; the rules associated to regions are employed in a non-deterministic and maximally parallel manner; the rules involving both transformation and communication are responsible for evolving the current objects and transfer them among regions according to some targets; the output region will contain the result of the computation [13].

2.3 Ant Colony Optimization

Ant colony optimizations, ACO for short, a successful evolutionary paradigm of swarm intelligence inspired from the social behaviors of insects and of other animals [19], was initiated by Dorigo in the early 1990s to solve various combinatorial optimization problems [5]. ACO is inspired by the behavior of real foraging ants, which employ pheromone trails to mark their paths to food resources. The main ACO algorithms in the literature include the earliest ant system, MAX-MIN ant system, rank-based ant system, hyper-cube ant system, and ant colony system (ACS). According to the studies in [5, 19, 20, 24], the ACS is one of the most powerful ACO algorithms. Therefore, we consider the use of ACS to construct the ACOPS. In what follows the TSP is taken as an example to describe the ACS.

In the ACS, the TSP is mapped onto a graph called a construction graph in such a manner that feasible solutions of the problem correspond to paths on the construction graph. Artificial ants successively produce better feasible solutions by modifying pheromones deposited on the edges of the TSP graph. The pseudocode algorithm for ACS is shown in Fig. 2, where each step is described as follows.

(i) In this step, the pheromone values are initialized to a value τ_0 ($\tau_0 = 1/ND$) at step $t = 0$, where N is the number of cities in a TSP and D is an arbitrary solution.

(ii) In the *While* loop, M represents the number of ants. Initially the M ants are randomly placed in the N nodes of the TSP graph as the initial state of a tour construction. Each ant uses a pseudorandom proportional rule to choose the next city it will visit. For instance, the kth ant in the ith city chooses the next city j by using the following formula

$$j = \begin{cases} \arg\max\limits_{l \in \mathcal{N}_i^k} \{[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}\}, & \text{if } q \le q_0 \\ J, & \text{otherwise} \end{cases} \quad (1)$$

where τ_{il} is the pheromone value of the edge connecting the ith node and the lth node; η_{il} is a heuristic information value; the parameters α and $\beta(\alpha > 0$

```
        Begin
            $t \leftarrow 1$
(i)         Initialize pheromone values
            While (not termination condition) do
                For $k$=1, 2, … , $M$
                    For $n$=1, 2, … , $N$
(ii)                    Construct a tour
(iii)                   Local pheromone update
                    End For
                End For
(iv)            Global pheromone update
                $t \leftarrow t+1$
            End While
        End Begin
```

Fig. 2. Pseudocode algorithm for ACS

and $\beta > 0$) determine the relative importance of the pheromone value τ_{il} and the heuristic information η_{il}; $\mathcal{N}_i^k$ ($\mathcal{N}_i^k \subseteq \mathcal{N}$) is the set of all nodes that the kth ant in the ith city can visit, where $\mathcal{N}$ is the set of all the nodes in the TSP graph; $q_0(0 \leq q_0 \leq 1)$ is a user-defined parameter specifying the distribution ratio of the two choices; q is a random number generated by using a uniform distribution function in the interval [0, 1]; J means that the next city j is chosen by using a random proportional rule, i.e., the kth ant in the ith city visits the city j at the next step according to the probability

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum\limits_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, & j \in \mathcal{N}_i^k \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

(iii) After an ant constructs a tour, it will update the pheromone value τ_{ij} of the tour by applying a rule as follows

$$\tau_{ij} = (1 - \upsilon)\tau_{ij} + \upsilon\tau_0 \tag{3}$$

where $\upsilon(0 < \upsilon < 1)$ is a local pheromone decay coefficient. The local pheromone update is used to encourage subsequent ants to choose other edges and, hence, to produce different solutions, by decreasing the pheromone value on the traversed edges. Thus, this step is helpful to the exploration of more solutions.

(iv) This step is to update the pheromone values of all the edges in the TSP graph by employing the best solution searched. To be specific, the pheromone value $\tau_{ij}(t)$ of the edge connecting the ith node and the jth node at generation t is modified as the pheromone value $\tau_{ij}(t+1)$ at generation $t+1$, i.e.,

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{4}$$

where $\rho (0 < \rho \leq 1)$ is a global pheromone decay coefficient and is also called the evaporation rate of the pheromone, and $\Delta\tau_{ij}(t)$ is

$$\Delta\tau_{ij}(t) = \begin{cases} 1/D_b, & \text{if } (i,j) \in T_b \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where D_b is the minimal distance searched, that is, the best solution searched, and T_b is the shortest path corresponding to D_b. The global pheromone update is to guide ants toward the best path searched.

2.4 ACOPS

In this subsection, we use the hierarchical framework of cell-like P systems and its evolution rules in a slightly modified way, and the parameterized probabilistic model, i.e., the pheromone model, of ACO, to specify the ACOPS algorithm. More specifically, the ACOPS applies the OLMS [13] to organize objects and evolution rules. The objects consist of ants or TSP construction graphs. The evolution rules, which are responsible to evolve the system and select the best ant, include a tour construction, and transformation/communication rules implemented by using local and global pheromone update rules.

More precisely the P system-like framework will consist of:

(i) a membrane structure $\mu = [_0[_1]_1, [_2]_2, \cdots, [_m]_m]_0$ with $m+1$ regions delimited by m elementary membranes and the skin membrane denoted by 0;
(ii) a vocabulary that consists of all the ants;
(iii) a set of terminal symbols, T, TSP construction graphs;
(iv) initial multisets $w_0 = \lambda$,
$w_1 = A_1 A_2 \cdots A_{M_1}$,
$w_2 = A_{M_1+1} A_{M_1+2} \cdots A_{M_2}$,
$\cdots\cdots$
$w_m = A_{M_{(m-1)}+1} A_{M_{(m-1)}+2} \cdots A_{M_m}$
where $A_i, 1 \leq i \leq M$, is an ant; M_j, $1 \leq j \leq m$, is the number of ants in the w_j; $\sum_{j=1}^{m} M_j = M$, where M is the total number of ants in this computation;
(v) rules which are categorized as
  a) tour construction rules in each of the compartments 0 to m; these are transformation-like rules which construct tours for the ants (see (ii) in the ACS presentation);
  b) communication rules which use pheromone values to update the edges of the TSP graphs. There are three levels of communication rules. The first level corresponding to the local pheromone update strategy of the ACS is utilized to exchange information between the current ant and its subsequent ant. The second and third levels of communication rules come from the global pheromone update strategy in the ACS. The second one implements information exchange between the best ant and the rest

within a certain membrane. The third one performs the communication between the ants in the elementary membranes and those in the skin membrane.

In the ACOPS the initial colony of ants is scattered across the membrane structure. The initial colony will consist of the multisets $w_1, \cdots, w_m$. Each of the ants in the elementary membranes uses the rules of type (a) to sequentially construct its tours. Going through N(the number of cities) steps, an ant will sketch a whole path for the N cities. If all the ants have their paths, the current generation is assessed compartment by compartment to select the best fit ant for each elementary membrane. The best ant is used to adjust the pheromone values in the TSP graph to communicate with the other ants in the same elementary membrane. Every $g_i (i = 1, 2, \cdots, m)$ generations for the ith compartment, the best ant is sent out to the skin membrane. Thus m ants from m elementary membranes form the initial objects in the skin membrane. These ants evolve independently g_0 generations to elect a best one to communicate with the ants in each elementary membrane. The process will stop according to a preset termination condition, such as a certain number of iterations. The pseudocode algorithm for the ACOPS is summarized in Fig. 3, where each step is described as follows.

```
        Begin
          $t \leftarrow 1$
(i)       Initialize the membrane structure
          While (not termination condition) do
(ii)          Scatter ants into elementary membranes
(iii)         Determine iterations for each of elementary membranes
              For $i$ =1, 2, .., $m$
(iv)              Perform ACS inside the $i$th elementary membrane
              End
(v)           Form a colony of ants in the skin membrane
(vi)          Perform ACS in the skin membrane
(vii)         Execute global communication
              $t \leftarrow t+1$
          End
        End
```

Fig. 3. Pseudocode algorithm for the ACOPS

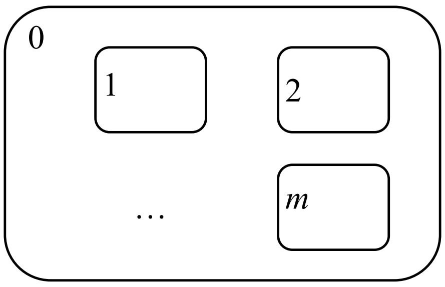


Fig. 4. The one level membrane structure

(i) In this step, the OLMS, shown in Fig. 4, is constructed. How to choose the parameter m will be discussed in Section 3.

(ii) The M ants forming a colony are scattered across the m elementary membranes in such a random way that guarantees each elementary membrane contains at least two ants. This is helpful to perform the second level of the communication process. Thus, the number of ants in an elementary membrane varies from 2 to $M - 2m + 2$.

(iii) This step determines the number of iterations for each elementary membrane to independently perform ACS. To be specific, the number $g_i (i = 1, 2, \cdots, m)$ of iterations for the ith elementary membrane is generated randomly between $g_{\min}$ and $g_{\max}$, i.e.,

$$g_i = g_{\min} + \lfloor \text{rand}(0,1) \cdot (g_{\max} - g_{\min}) \rfloor \tag{6}$$

where $g_{\min}$ and $g_{\max}$ are lower and upper limits of iterations for elementary membranes, respectively; $\lfloor \cdot \rfloor$ is a function rounding an element to the nearest smaller integer.

(iv) In each of the m elementary membranes, the ACS algorithm shown in Fig. 2 is performed independently, i.e., the tour construction, local pheromone update and global pheromone update are sequentially carried out for $g_i (i = 1, 2, \cdots, m)$ iterations.

(v) The colony of ants in the skin membrane is formed by using the best ants of elementary membranes. Each compartment sends the best ant out into the skin membrane and therefore there are m ants in total.

(vi) The ACS algorithm shown in Fig. 2 is performed independently in the skin membrane for g_0 iterations as in step (iv). The parameter g_0 is determined using (6).

(vii) The global communication is used to exchange some information between the ants in the skin membrane and those in the elementary membranes. To be specific, the best one ant in the skin membrane is employed to update the pheromone values of the TSP graph in each of elementary membranes. This operation has a positive effect on the ants in the compartments toward better fitness, the shorter path.

3 Experiments and Results

The performance of the ACOPS is tested on various TSP instances. First of all, it is discussed how to set the number of elementary membranes by using 4 TSP benchmarks. Then 20 benchmarks are applied to compare ACOPS and its counterpart ACO algorithm. Subsequently, experimental comparisons between ACOPS and Nishida's algorithms are performed on 8 benchmark TSP problems. In these experiments, several parametric and non-parametric tests are employed to analyze the ACOPS behavior.

3.1 Parameter Setting

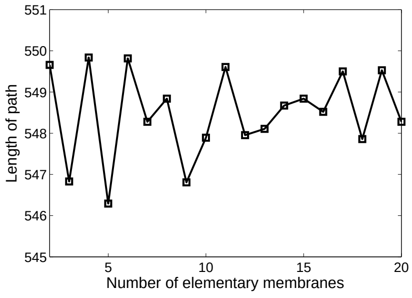
This subsection uses four TSP benchmarks, Eil76, Eil101, Ch130 and Ch150, to discuss how to choose the number m of elementary membranes and the number $g_i (i = 1, 2, \cdots, m)$of generations for each elementary membrane. The four TSPs have N =76, 101, 130 and 150 cities, respectively. According to the studies in the literature, the parameters in the experiments are chosen as follows: $M = 40$, $\alpha = 1$, $\beta = 3$, $\rho = 0.6$, $\upsilon = 0.1$ and $q_0 = 0.9$. We use the number 10000 of function evaluations as the termination criterion for all tests.

We first investigate the effect of the number of elementary membranes on the ACOPS performance. On the basis of the ACOPS description, the parameter m varies from 2 to 20. The parameters $g_{\min}$ and $g_{\max}$ are set to 10 and 30, respectively. The performances of the ACOPS for each of the 19 cases are evaluated by using the best solutions and their corresponding elapsed time per run, and the mean of best solutions and their corresponding mean of elapsed time per run, of 20 independent runs. Experimental results are shown in Fig. 5 – Fig. 12.

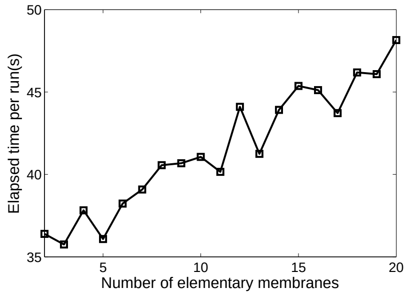
It can be seen from Fig. 5 – Fig. 12 that there are some general trends. Both the best solutions and the mean of best solutions over 20 runs have fluctuant behavior. To be specific, there is a first rapid fall and then several waves of higher values follow. The elapsed time per run has a general increase as m goes up from 2 to 20. It is worth pointing out that the general trends become clearer as the complexity of the problem increases. From these experimental results, a trade-off value for the parameter m between the quality of solutions and the elapsed time could be about 4.

In what follows we set the number of elementary membranes to 4 to conduct a further investigation on the effects of the number of communications (NoC) between the skin membrane and the elementary membranes, i.e., the number of global communications, on the ACOPS performance. Let the NoC vary from 1 to 40. The number of function evaluations (NoFE) as the stopping criterion is 10000. The parameter $g_{\min}$ is set 10. Thus, according to the ACOPS description, the $g_{\max}$ can be obtained from the following formula

$$g_{\max} = \frac{2 \cdot \text{NoFE}}{\text{NoC} \cdot (N + m)} - g_{\min} \tag{7}$$

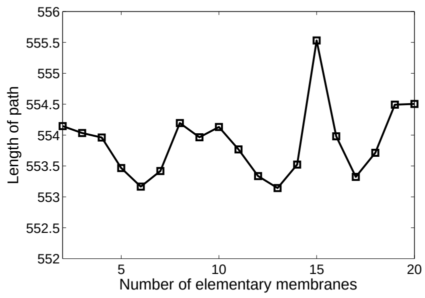


(a) Best solutions

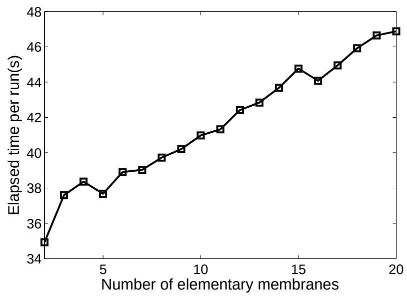


(b) Elapsed time per run

Fig. 5. Experimental results of Eil76 with different membranes

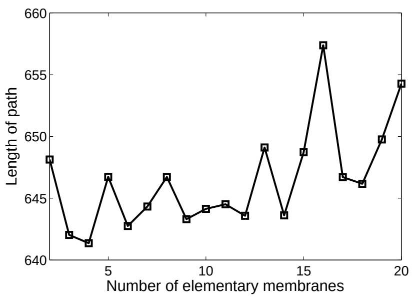


(a) Mean of best solutions

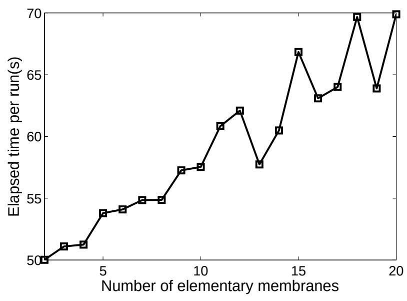


(b) Mean of elapsed time per run

Fig. 6. Experimental results of Eil76 with different membranes



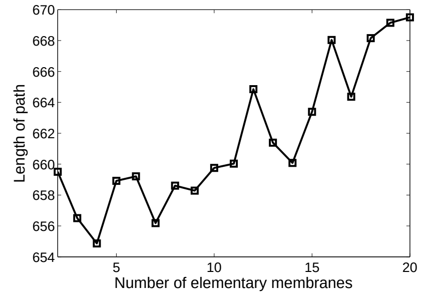
(a) Best solutions



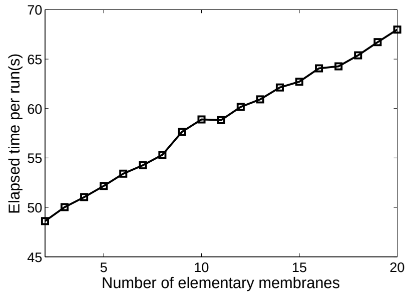
(b) Elapsed time per run

Fig. 7. Experimental results of Eil101 with different membranes

where N and m are the total number of ants and the number of elementary membranes (also is the number of ants in the skin membrane), respectively. We also use the four TSP benchmarks, Eil76, Eil101, Ch130 and Ch150, to carry out the

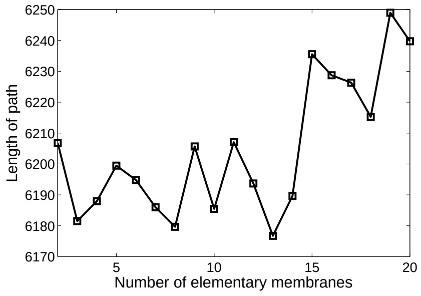


(a) Mean of best solutions

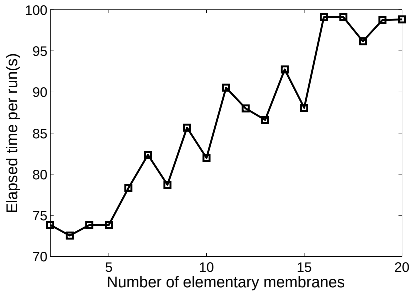


(b) Mean of elapsed time per run

Fig. 8. Experimental results of Eil101 with different membranes

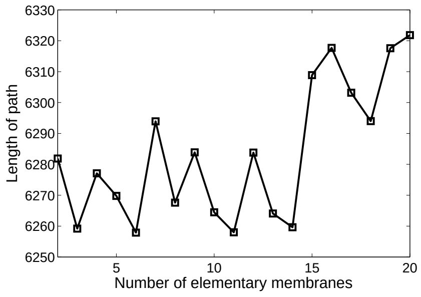


(a) Best solutions

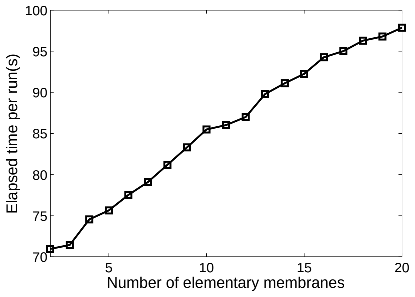


(b) Elapsed time per run

Fig. 9. Experimental results of Ch130 with different membranes



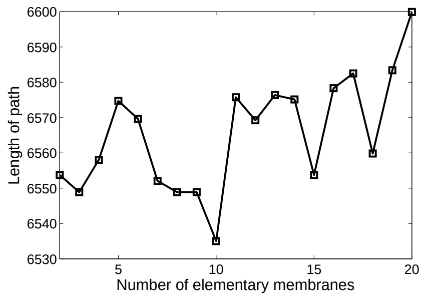
(a) Mean of best solutions



(b) Mean of elapsed time per run

Fig. 10. Experimental results of Ch130 with different membranes

experiments. For each of the 40 cases, we record the best solutions and their corresponding mean of elapsed time per run, and the mean of best solutions and their corresponding mean of elapsed time per run, to assess the ACOPS performance.


(a) Best solutions

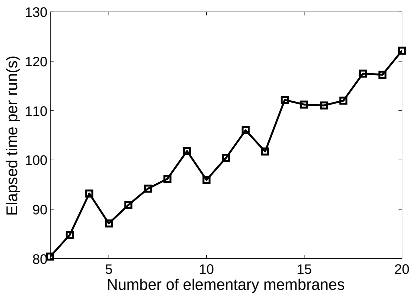
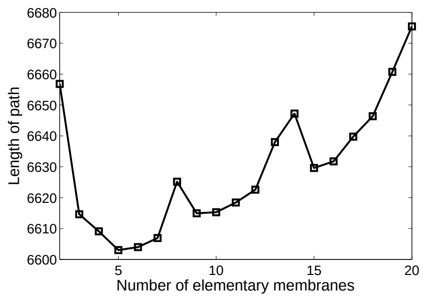

(b) Elapsed time per run

Fig. 11. Experimental results of Ch150 with different membranes


(a) Mean of best solutions

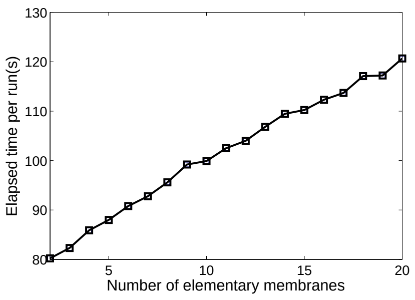

(b) Mean of elapsed time per run

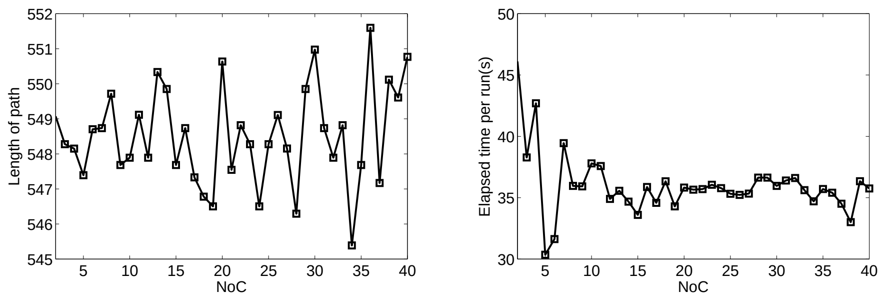
Fig. 12. Experimental results of Ch150 with different membranes

The independent runs for each case are 20. Experimental results are shown in Fig. 13 – Fig. 20.

Figures 13 – 20 show that the best solutions and the mean of best solutions over 20 runs have a behavior oscillating between various maxima and minima. The elapsed time per run goes through a drastic fluctuation and then stays a relatively steady level. We note that the trends become clearer as the complexity of the problem increases. Considering a trade-off between quality of solutions and the elapsed time, the recommended value for the NoC could be chosen in the range [15, 35]. Thus, given a certain value of NoFE, an appropriate value for the parameter gmax could be determined in an interval, according to (7).

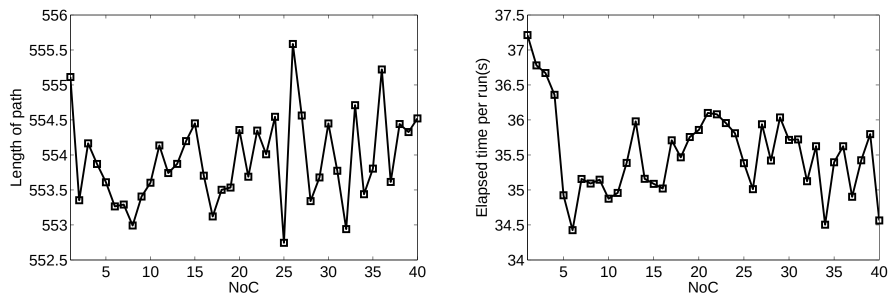
3.2 Comparisons with ACO and Statistical Analysis

To draw a comparison between the ACOPS and its counterpart ACO, we use 20 symmetric TSP benchmark problems to conduct experiments. The test problems, shown in Table 1, were chosen either because there were data available online in the literature, or the optimal solutions are known. They are challenging enough



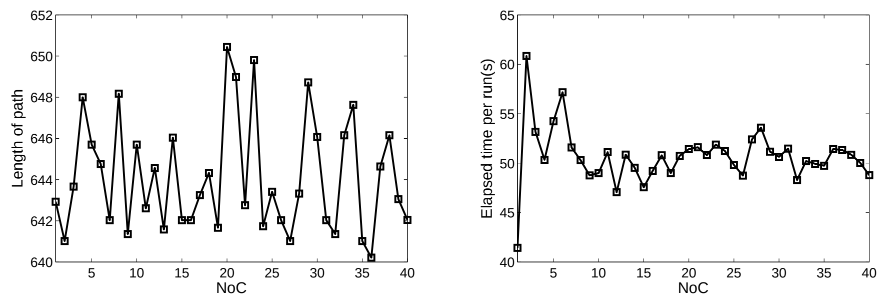
(a) Best solutions (b) Elapsed time per run

Fig. 13. Experimental results of Eil76 with NoC



(a) Mean of best solutions (b) Mean of elapsed time per run

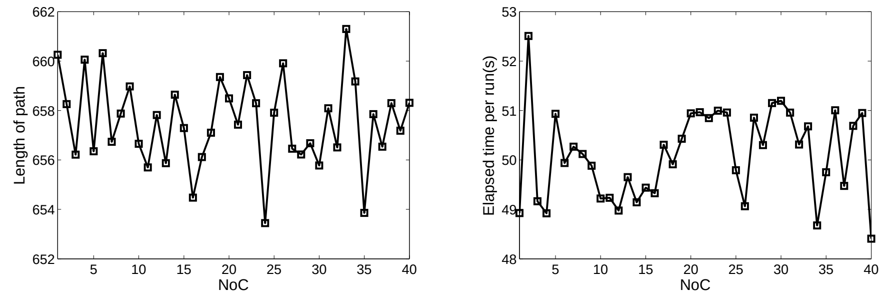
Fig. 14. Experimental results of Eil76 with NoC



(a) Best solutions (b) Elapsed time per run

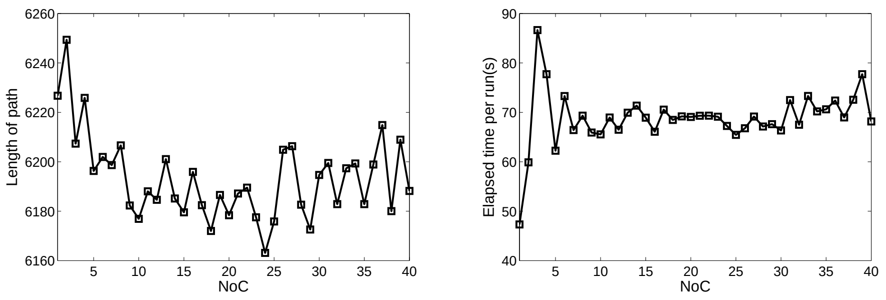
Fig. 15. Experimental results of Eil101 with NoC

for making fair comparisons between the two algorithms, in terms of solving difficult instances of TSPs. In the following experiments, the ACOPS and ACO have identical settings for parameters: $M = 40$, $\alpha = 1$, $\beta = 3$, $\rho = 0.6$, $\upsilon = 0.1$, $q_0 = 0.9$



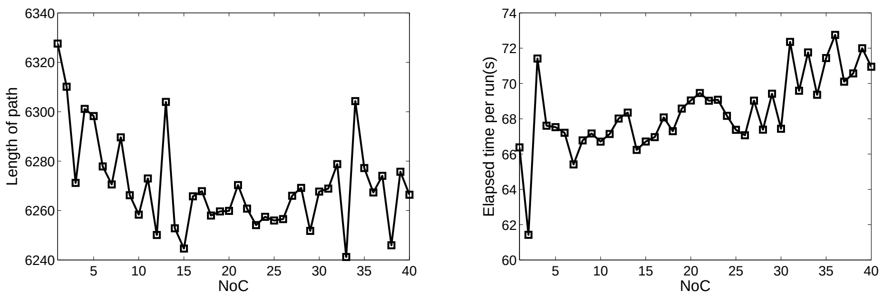
(a) Mean of best solutions (b) Mean of elapsed time per run

Fig. 16. Experimental results of Eil101 with NoC



(a) Best solutions (b) Elapsed time per run

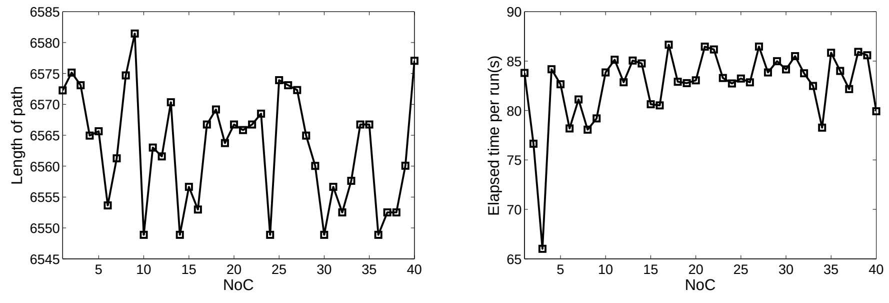
Fig. 17. Experimental results of Ch130 with NoC



(a) Mean of best solutions (b) Mean of elapsed time per run

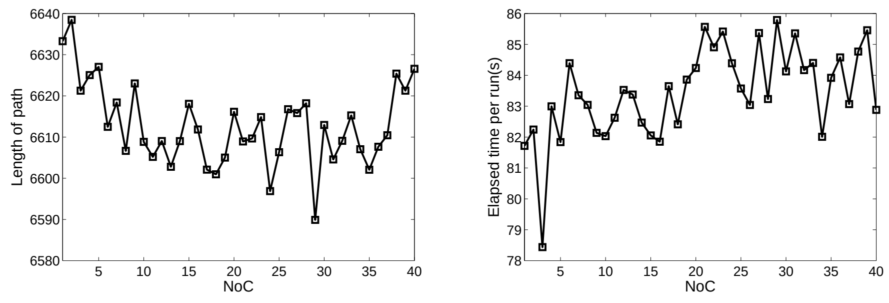
Fig. 18. Experimental results of Ch130 with NoC

and the NoFE, also listed in Table 1, for different TSPs as the termination criterion. Additionally, in the ACOPS, the number of elementary membranes, the parameters $g_{\min}$ and $g_{\max}$ are set to 4, 10 and 30, respectively. The performances



(a) Best solutions (b) Elapsed time per run

Fig. 19. Experimental results of Ch150 with NoC



(a) Mean of best solutions (b) Mean of elapsed time per run

Fig. 20. Experimental results of Ch150 with NoC

of the ACOPS and ACO are evaluated by using the following statistical results over 20 independent runs: the best, the worst and the average length of paths. Experimental results are listed in Table 1.

As shown in Table 1, the ACOPS achieves better results than the ACO in 19 out of 20 instances, in terms of the best and mean solutions. We go further to apply statistical techniques to analyze the behavior of the two algorithms, ACOPS and ACO, over the 20 TSPs. Parametric and non-parametric approaches are two main ways of statistical methods [25]. The parametric approach, also called single-problem analysis, employs a parametric statistical analysis t-test to check whether there is a significant difference between two algorithms applied to an optimization problem. The non-parametric approach, also called multiple-problem analysis, utilizes non-parametric statistical tests, such as Wilcoxon's and Friedman's tests, to compare different algorithms whose results represent average values for each problem, regardless of the inexistence of relationships among them. Thus, a 95% confidence Student t-test is first used to check whether there are significant differences between ACOPS and ACO. Two non-parametric tests, Wilcoxon's and

Table 1. A comparison between ACOPS and ACO ('+' and '-' represent significant difference and no significant difference, respectively. '—' means no optimum available)

TSP	NoFE	ACO			ACOPS			*t*-test	Optimum
		Best	Average	Worst	Best	Average	Worst		
ulysses16	1e+4	74.11	74.11	74.11	73.99	74.02	74.23	3.47e-6(+)	74
att48	2e+4	33588.34	33654.16	33740.35	33523.71	33644.97	34060.49	7.05e-1(-)	33524
eil76	3e+4	545.39	546.22	551.93	544.37	551.62	555.55	3.48e-7(+)	538
kroA100	4e+4	21577.69	21776.91	22320.91	21285.44	21365.64	21552.00	4.00e-8(+)	21282
eil101	4e+4	642.66	652.30	684.19	640.98	648.48	664.24	2.03e-1(-)	629
lin105	4e+4	14383.00	14472.38	14482.31	14383.00	14444.77	14612.43	8.61e-2(+)	14379
ch130	4.5e+4	6204.09	6268.43	6333.16	6148.99	6205.54	6353.69	1.02e-3(+)	6110
gr137	4.5e+4	718.92	725.02	749.93	709.91	718.85	738.35	6.63e-3(+)	—
pr144	5e+4	58587.14	58612.82	58687.80	58535.22	58596.00	58761.43	2.24e-1(-)	58537
ch150	5e+4	6595.00	6630.59	6689.79	6548.89	6570.86	6612.46	1.05e-7(+)	6528
rat195	6e+4	2370.24	2392.69	2434.39	2348.32	2355.23	2373.79	1.61e-7(+)	2323
d198	6e+4	16172.77	16266.93	16530.79	16073.13	16192.89	16381.91	3.75e-2(+)	15780
kroa200	6e+4	29597.01	29988.74	30466.71	29453.10	29552.92	29688.13	1.92e-6(+)	29437
gr202	6e+4	496.48	496.96	499.53	488.41	494.21	499.44	6.82e-4(+)	—
tsp225	7e+4	4067.96	4146.32	4262.76	3904.46	3971.68	4044.32	2.11e-9(+)	3916
gr229	7e+4	1739.77	1763.80	1802.44	1725.84	1756.28	1792.91	2.11e-1(-)	—
gil262	8e+4	2452.82	2487.58	2512.85	2407.68	2431.58	2450.65	4.86e-10(+)	2378
a280	9e+4	2626.44	2683.21	2787.61	2595.31	2636.49	2728.06	8.46e-4(+)	2579
pr299	10e+4	51050.78	52103.27	53698.23	49370.69	51021.74	52251.21	7.69e-4(+)	48191
lin318	10e+4	44058.08	45297.99	46410.50	42772.12	43433.54	45194.62	5.95e-9(+)	42029

Friedman's tests, are applied to check whether the two algorithms are significantly different or not. The level of significance considered is 0.05. Results of *t*-test are listed in Table 1. Results of Wilcoxon's and Friedman's tests are shown in Table 2. In Table 1 and 2, the symbols '+' and '-' represent significant difference and no significant difference, respectively. The *t*-test results in Table 1 demonstrate that there are 16 significant differences between the two algorithms. The *p*-values of the two non-parametric tests in Table 2 are far smaller than the level of significance 0.05, which indicates that the ACOPS really outperforms the ACO. It is worth noting that the study in [25] shows that the non-parametric statistical tests are more appropriate than parametric statistical tests in the analysis of the behavior of optimization algorithms over multiple optimization problems.

Table 2. Results of non-parametric statistical tests for ACOPS and ACO in Table 1. '+' represents significant difference

Tests	ACOPS vs ACO
Wilcoxon test(*p*-value)	1.6286e-004(+)
Friedman test(*p*-value)	5.6994e-005(+)

3.3 Comparisons with Nishida's Algorithms and Statistical Analysis

In [6, 7, 8], Nishida proposed a membrane algorithm combining an NMS P systems structure and a local search. Eight TSP benchmarks were applied to test the performances of the algorithm and its variants. In his experiments, the NMS with 2, 10, 30, 50, 70 and 100 membranes, respectively, was discussed. The maximal number of iterations, which can be equivalent to a certain number of function evaluations (NoFE), was used as the termination criterion. The number of trials was

20. The ACOPS algorithm stops according to a prescribed NoFE. The parameters of the ACOPS are assigned as follows: $M = 40$, $\alpha = 1$, $\beta = 3$, $\rho = 0.6$, $\upsilon = 0.1$, $q_0 = 0.9$, $m = 4$, $g_{\min} = 10$ and $g_{\max} = 30$. Experimental comparisons between the ACOPS and the Nishida's algorithm are listed in Tables 3-5, in which NoM represents the number of membranes. The results of the ACOPS are obtained from 20 independent runs. In Tables 3 and 4, the results of Nishida's algorithm are obtained from [7, 8] and the NoFE for each cases are calculated according to the number of iteration and the number of membranes. The performance of the ACOPS is also tested on the Eil51 and KroA100 TSPs with different NoFE. Table 5 lists the experimental results of Nishida's algorithm and the equivalent NoFE for each of the 8 TSP instances calculated by using 50 membranes and 100000 iterations. The results of Wilcoxon's and Friedman's tests are given in Table 6.

Table 3. Comparisons of Nishida's algorithm and ACOPS with Eil51 TSP

	Nishida's algorithm					**ACOPS**				
NoM	2	10	30	50	70	4				
NoFE	1.2e+5	7.6e+5	2.36e+6	3.96e+6	5.56e+6	1e+4	2e+4	3e+4	4e+4	5e+4
Best	440	437	432	429	429	429.4841	429.4841	428.9816	428.9816	428.9816
Worst	786	466	451	444	443	435.5985	436.3928	434.9739	433.6050	433.8558
Average	522	449	441	435	434	432.3858	431.8023	431.3146	430.5506	430.4495

Table 4. Comparisons of Nishida's algorithm and ACOPS with KroA100 TSP

	Nishida's algorithm						**ACOPS**					
NoM	2	10	30	50	70	100	4					
NoFE	3e+5	1.9e+6	5.9e+6	9.9e+6	1.39e+7	1.99e+7	1e+4	2e+4	4e+4	6e+4	8e+4	1e+5
Best	23564	21776	21770	21651	21544	21299	21331	21285	21285	21285	21285	21285
Worst	82756	24862	23940	24531	23569	22954	22332	21665	21552	21475	21427	21575
Average	34601	23195	22878	22590	22275	21941	21593	21407	21367	21337	21320	21362

Table 5. Comparisons of Nishida's algorithm and ACOPS with 8 TSPs

TSP	**Nishida's algorithm**				**ACOPS**			
	NoFE	Best	Average	Worst	NoFE	Best	Average	Worst
ulysses22	9.9e+7	75.31	75.31	75.31	2e+4	75.31	75.32	75.53
eil51	9.9e+7	429	434	444	4e+4	429	431	434
eil76	9.9e+7	556	564	575	6e+4	546	551	558
eil101	9.9e+7	669	684	693	8e+4	641	647	655
kroA100	9.9e+7	21651	22590	24531	1e+5	21285	21320	21427
ch150	9.9e+7	7073	7320	7633	1.2e+5	6534	6560	6584
gr202	9.9e+7	509.7	520.1	528.4	1.4e+5	489.2	492.7	497.1
tsp225	9.9e+7	4073.1	4153.6	4238.9	7e+4	3899.6	3938.2	4048.2

As compared with Nishida's algorithm, the ACOPS uses much smaller NoFE to achieve better solutions, which is shown in Table 3-5. Small NoFE means low computing complexity. The non-parametric statistical analysis shows that the two algorithms have also a significant difference.

Table 6. Results of non-parametric statistical tests for Nishida's algorithm and ACOPS in Table 5. '+' represents significant difference

Tests	**Nishida's algorithm vs ACOPS**
Wilcoxon test(p-value)	0.0156 (+)
Friedman test(p-value)	0.0339 (+)

4 Conclusions

This work is the first attempt to discuss the interaction between P systems and ant colony optimization. We present an approximate optimization algorithm combining the hierarchical structure of compartments and communication/transformation evolution rules of P systems, and the pheromone model of ant colony optimization. The introduced approach is used to solve the well-known and extensively studied NP-hard problem, traveling salesman problem. The better optimization performance of the ACOPS is verified by comparing it with its counterpart ACO and Nishida's algorithms. In order to thoroughly test the capabilities of this approach, our future studies will focus on the use of the ACOPS to producing solutions to some real-world engineering problems.

References

1. Păun, Gh.: Computing with Membranes, *Journal of Computer and System Sciences*, **61**(1), 2000, 108-143.
2. Păun, Gh., Rozenberg, G., Salomaa, G.:(Eds.), *Handbook of membrane computing.* Oxford: Oxford University Press, 2009.
3. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems, *Fundamenta Informaticae*, **71**(2-3), 2006, 279-308.
4. Zhang, G. X., Liu, C. X., Gheorghe, M., Ipate, F.: Solving Satisfiability Problems with Membrane Algorithm, *Proc. of the 4th International Conference on Bio-Inspired Computing: Theories and Applications*, 2009, 29–36.
5. Blum, C.: Ant colony optimization Introduction and recent trends, *Physics of Life Reviews*, **2**(4), 2005, 353-373.
6. Nishida, T. Y.: An application of P-system: A new algorithm for NP-complete optimization problems, *Proc. 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, Orlando, 2004, 109-112.
7. Nishida, T. Y.: Membrane algorithms: Approximate algorithms for NP-complete optimization problems, *Applications of Membrane Computing*, 2006, 303-314.
8. Nishida, T. Y.: Membrane Algorithm: An Approximate Algorithm for NP-Complete Optimization Problems Exploiting P-Systems", *Proc. 6th International Workshop on Membrane Computing*, Vienna, 2005, 26-43.

9. Leporati ,A., Pagani, D.: A membrane algorithm for the min storage problem, *Proc. of Workshop on Membrane Computing*, **4361** Berlin: Springer, 2006, 443-462.
10. Huang, L., He, X. X., Wang, N., Xie, Y.: P Systems Based Multi-objective Optimization Algorithm, *Progress in Natural Science*, **17**(1), 2007, 458–465.
11. Huang, L., Wang, N.: An optimization algorithm inspired by membrane computing, *Proc. of ICNC*, **4222**, 2006, 49-52.
12. Zaharie, D., Ciobanu, G.: Distributed Evolutionary Algorithms Inspired by Membranes in Solving Continuous Optimization Problems, *Proc. of the WMC*, **LNCS 4361**, 2006, 536–553.
13. Zhang, G. X., Gheorghe, M., Wu, C. Z.: A Quantum-Inspired Evolutionary Algorithm Based on P Systems for Knapsack Problem *Fundamenta Informaticae*, **87**(1), 2008, 93–116.
14. Liu, C. X., Zhang, G. X., Zhu, Y. H., Fang, C., Liu, H. W.: A Quantum-inspired evolutionary algorithm based on P systems for radar emitter signals, *the Fourth International Conference on Bio-Inspired Computing: Theories and Applications*, Beijing, 2009, 24-28.
15. Liu, C. X., Zhang, G. X., Liu, L. W., Gheorghe, M., Ipate, F.: An Improved Membrane Algorithm for Solving Time-Frequency Atom Decomposition, *WMC 2009, LNCS*, **5957**, Gh. Păun, Ed.: Springer, 2010, 371–384.
16. Liu, C. X., Zhang, G. X., Liu, H. W.: A Memetic Algorithm Based on P Systems for IIR Digital Filter Design," *Proc. of the 8th IEEE International Conference on Pervasive Intelligence and Computing*, 2009, 330-334.
17. Huang, L., Suh, I. H.: Controller design for a marine diesel engine using membrane computing, *International Journal of Innovative Computing Information and Control*, **5**(4), 2009, 899-912.
18. Păun, G., Pérez-Jiménez, M. J.: Membrane computing: Brief introduction, recent results and applications, *Biosystems*, **85**(1), 2006, 11-22, 2006.
19. Dorigo, M., Birattari, M., Stützle, T.: Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique, *IEEE Computational Intelligence Magazine*, **1**(4), 2006, 28-39.
20. Dorigo, M., Blum, C.: Ant colony optimization theory: a survey, *Theoretical Computer Science*, **344**(2-3), 2005, 243-278.
21. Uğur, A., Aydin, D.: An interactive simulation and analysis software for solving TSP using Ant Colony Optimization algorithms, *Advances in Engineering Software*, **40**(5), 2009, 341-349.
22. Păun, Gh.: Tracing some open problems in membrane computing, *Romanian Journal of Information Science and Technology*, **10**(4), 2007, 303-314.
23. Păun, Gh., Rozenberg, G.: A guide to membrane computing, *Theor. Comput. Sci.*, **287**(1), 2002, 73–100.
24. Wu, Z., Zhao, N., Ren, G., Quan, T.: Population declining ant colony optimization algorithm and its applications, *Expert Systems with Applications*, **36**(3), 2009, 6276-6281.
25. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization, *J. Heuristics.* 10.1007/s10732 -008-9080-4.

Author Index