Tenth Brainstorming Week on Membrane Computing

Sevilla, January 30 – February 3, 2012

Volume I

Miguel A. Martínez-del-Amor
Gheorghe Păun
Ignacio Pérez-Hurtado
Francisco J. Romero-Campero
Editors

Tenth Brainstorming Week on Membrane Computing

Sevilla, January 30 – February 3, 2012

Volume I

Miguel Ángel Martínez del Amor
Gheorghe Păun
Ignacio Pérez Hurtado de Mendoza
Francisco José Romero Campero
Editors

RGNC REPORT 1/2012

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2012

ISBN: ??????
Depósito Legal: SE-????–06
Edita: Fénix Editora
Avda. de Cádiz, 7 – 1C
41004 Sevilla
fenixeditora@telefonica.net
Telf. 954 41 29 91

Preface

These proceedings, consisting in two volumes, contain the papers emerged from the Tenth Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from January 30 to February 3, 2012, 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 all the next editions took place in Sevilla at the beginning of February, each year.

The 2012 edition of BWMC was organized in conjunction with the First International Conference on Developments in Membrane Computing (ICDMC2012).

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

Slightly different from the previous meetings was the combination with the ICDMC2012, in the sense that several talks also had the style of a conference: more time dedicated to presenting achievements obtained by the research groups from where the participants came from, but also converging towards the style of the brainstorming, i.e., presenting frontier results, research topics, ongoing applications.

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

Selections of the papers from these volumes will be considered for publication in special issues of *Theoretical Computer Science* and of *International Journal of Computer Mathematics*.

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

- BWMC 2003: *Natural Computing* – volume 2, number 3, 2003, and *New Generation Computing* – volume 22, number 4, 2004;
- BWMC 2004: *Journal of Universal Computer Science* – volume 10, number 5, 2004, and *Soft Computing* – volume 9, number 5, 2005;
- BWMC 2005: *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006);
- BWMC 2006: *Theoretical Computer Science* – volume 372, numbers 2-3, 2007;
- BWMC 2007: *International Journal of Unconventional Computing* – volume 5, number 5, 2009;
- BWMC 2008: *Fundamenta Informaticae* – volume 87, number 1, 2008;
- BWMC 2009: *International Journal of Computers, Control and Communication* – volume 4, number 3, 2009;
- BWMC 2010: *Romanian Journal of Information Science and Technology* – volume 13, number 2, 2010;
- BWMC 2011: *International Journal of Natural Computing Research* – volume 2, numbers 2-3, 2011.

Other papers elaborated during the tenth 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. Artiom Alhazov, University of Milano - Bicocca, Italy, *aartiom@yahoo.com*
2. Ioan Ardelean, Institute of Biology of the Romanian Academy, Bucharest, Romania, *ioan.ardelean57@yahoo.com*
3. Mari Angels Colomer Cugat, University of Lleida, Spain, *colomer@matematica.udl.cat*
4. Erzsébet Csuhaj-Varjú, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary, *csuhaj@inf.elte.hu*
5. Rudolf Freund, Technological University of Vienna, Austria, *rudifreund@gmx.at*
6. Manuel García-Quismondo Fernández, University of Seville, Spain, *mgarciaquismondo@us.es*
7. Marian Gheorghe, University of Sheffield, United Kingdom, *m.gheorghe@sheffield.ac.uk*
8. Carmen Graciani Díaz, University of Seville, Spain, *cgdiaz@us.es*

9. Miguel A. Gutiérrez Naranjo, University of Seville, Spain, *magutier@us.es*
10. Florentin Ipate, University of Piteşti, Romania, *florentin.ipate@ifsoft.ro*
11. Jozef Kelemen, Silesian University, Opava, Czech Republic, *jozef.kelemen@fpf.slu.cz*
12. Abhay Krishna, CABIMER, Seville, Spain, *Abhay.Krishan@cabimer.es*
13. Raluca Lefticaru, University of Piteşti, Romania, *raluca.lefticaru@gmail.com*
14. Alberto Leporati, University of Milano - Bicocca, Italy, *leporati@disco.unimib.it*
15. Luis Felipe Macías Ramos, University of Seville, Spain, *lfmaciasr@us.es*
16. Vincenzo Manca, University of Verona, Italy, *vincenzo.manca@univr.it*
17. Luca Marchetti, University of Verona, Italy, *luca.marchetti@univr.it*
18. Miguel A. Martínez del Amor, University of Seville, Spain, *mdelamor@us.es*
19. Giancarlo Mauri, University of Milano - Bicocca, Italy, *mauri@disco.unimib.it*
20. Adam Obtulowicz, Polish Academy of sciences, Warsaw, Poland, *A.Obtulowicz@impan.pl*
21. Ana Brânduşa Pavel, Politehnica University of Bucharest, Romania, *anabrandusa@gmail.com*
22. Gheorghe Păun, Romanian Academy, Bucharest, Romania, and University of Seville, Spain, *gpaun@us.es*
23. Hong Peng, School of Mathematics and Computer Engineering, Xihua University, China, *ph.xhu@hotmail.com*
24. Ignacio Pérez Hurtado de Mendoza, University of Seville, Spain, *perezh@us.es*
25. Mario de J. Pérez Jiménez, University of Seville, Spain, *marper@us.es*
26. Antonio Enrico Porreca, University of Milano - Bicocca, Italy, *porreca@disco.unimib.it*
27. Raúl Reina Molina, University of Seville, Spain, *m75@gmail.com*
28. Agustín Riscos Núñez, University of Seville, Spain, *ariscosn@us.es*
29. Iurie Rogojin, Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova, Chişinău, Moldova, *yrogozhin@gmail.com*
30. Álvaro Romero Jiménez, University of Seville, Spain, *romero.alvaro@us.es*
31. Francisco José Romero Campero, University of Seville, Spain, *fran@us.es*
32. Jose María Sempere Luna, Polytechnical University of Valencia, Spain, *jsempere@dsic.upv.es*
33. Petr Sosík, Silesian University, Opava, Czech Republic, and Universidad Politécnica de Madrid, Spain, *petr.sosik@fpf.slu.cz*
34. Cristian Ştefan, University of Piteşti, Romania, *liviu.stefan@yahoo.com*
35. Luis Valencia Cabrera, University of Seville, Spain, *lvalencia@us.es*
36. György Vaszil, Faculty of Informatics, University of Debrecen, Hungary, *vaszil.gyorgy@inf.unideb.hu*
37. Serghei Verlan, University of Paris Est, France, *verlan@univ-paris12.fr*
38. Jun Wang, Electrical and Information Engineering, Xihua University, China, *wj.xhu@hotmail.com*
39. Claudio Zandron, University of Milano - Bicocca, Italy, *zandron@disco.unimib.it*

40. Gexiang Zhang, School of Electrical Engineering, Southwest Jiaotong University, China, *gexiangzhang@gmail.com*

As mentioned above, the meeting was organized by the Research Group on Natural Computing from Sevilla University (`http://www.gcn.us.es`)– and all the members of this group were enthusiastically involved in this (not always easy) work.

The meeting was supported from various sources: (i) Proyecto de Excelencia con investigador de reconocida valía, de la Junta de Andalucía, grant P08 – TIC 04200, (ii) Proyecto del Ministerio de Educación y Ciencia, grant TIN2009 – 13192, (iii) Instituto de Matemáticas de la Universidad de Sevilla (IMUS), (iv) Consejería de Innovacion, Ciencia y Empresas de la Junta de Andalucía, as well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Mario de Jesús Pérez-Jiménez
(Sevilla, May 3, 2012)

Contents

Contents of Volume II

Self-Stabilization in Membrane Systems

Artiom Alhazov[1,2], Marco Antoniotti[1], Rudolf Freund[3],
Alberto Leporati[1], Giancarlo Mauri[1]

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
E-mail: {artiom.alhazov,marco.antoniotti,
alberto.leporati,giancarlo.mauri}@unimib.it

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

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

Summary. In this paper we study a notion of self-stabilization, inspired from biology and engineering. Multiple variants of formalization of this notion are considered, and we discuss how such properties affect the computational power of multiset rewriting systems.

1 Introduction

Membrane systems, also called P systems, are a framework for (bioinspired) computational models, see [4], [5] and [7]. In this paper we consider a one-region rewriting model with symbol objects. In this case, membrane computing can be considered as (maximally parallel or sequential) multiset processing. In general, a computation is a sequence of transitions between configurations. Configurations are multisets, and the transitions are induced by rules, defined by reactants, products and control (additional applicability conditions, if any), viewed as formal computational systems (generating/accepting numeric/vector sets, or computing functions).

We will call a property dynamic if it depends on the behavior of a system and cannot be easily derived from its description (as opposed to syntactic properties). Given any finite computation, we assume that the property is easily verifiable. The two usual sources of undecidability are a) that we do not always know whether we are dealing with finite or infinite computations, and b) that some properties are defined on infinite number of computations (due to non-determinism, to the initial input or to some other parameter). In the case of this paper, another source of potential undecidability is the finite set to be reached as given in the definitions below.

Since in this paper we will deal with reachability issues, we would also like to mention the connection with temporal logic [2].

Self-stabilization is a known concept in conventional distributed computing, [8], as well as in systems biology, but as far as we know, it has not yet been considered in the framework of membrane computing. It has been recalled by Jacob Beal during the Twelfth Conference in Membrane Computing, CMC12, and an attempt to formalize it in the membrane computing framework has been done in [1]. The underlying idea is the tolerance of natural and engineering systems to perturbations. The formulation from [8] says:

A system is self-stabilizing if and only if:

1. Starting from any state, it is guaranteed that the system will eventually reach a correct state (convergence).
2. Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens (closure).

In case of inherently non-deterministic systems, "with probability 1" should be added. Based on this concept, we propose to consider a few formal properties, following the discussion below.

In this paper we consider fully cooperative multiset rewriting, possibly with promoters/inhibitors/priorities, operating either in the maximally parallel or the sequential mode. We consider a single working region only, for two reasons. First, the properties of interest are unaffected by flattening the static membrane structure. Second, we would currently like to avoid the discussion about reachability related to "arbitrary configurations" with dynamic membrane structure.

2 Definitions

We assume the reader to be familiar with the basics of formal language theory, e.g., we refer to [6].

An *alphabet* is a finite non-empty set V of abstract *symbols*. The free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. The family of all finite (recursive, recursively enumerable) sets of positive integers is denoted by $NFIN$ ($NREC$, NRE, respectively).

2.1 Membrane systems

A one-region (rewriting) membrane system is a tuple

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

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

A configuration of the system is represented by a multiset of objects from O contained in the region, and a rule $r : u \to v$ is applicable if the current configuration contains the multiset specified by u. Furthermore, applicability may be controlled by promoters ($r : u \to v|_a$), inhibitors ($r : u \to v|_{\neg b}$), or priorities($r' > r$). Throughout the paper, we will use the word *control* to mean that at least one of these three features is allowed. In such cases, in addition to the availability of u for a rule r to be applicable, the promoter a must be present in the current configuration, the inhibitor b has to be absent in the current configuration, and no rule r' with higher priority than r is allowed to be applicable, respectively.

A computation step in the sequential mode consists of the non-deterministic application of one applicable rule, replacing its left side with its right side. In the maximally parallel mode, multiple applicable rules have to be applied multiple times, to disjoint submultisets, in a non-deterministic way, possibly leaving some objects idle, under the condition that no further rule is applicable to them. The computation step is denoted by the binary relation $\Rightarrow$. A computation halts when no rule is applicable to the current configuration (halting configuration).

For a generating system, the result of a halting computation is the total number of objects in the system when it halts. The set of numbers generated by a P system is the set of results of its computations. An accepting system is described as (O, Σ, w, R), where Σ is an input alphabet: instead of w, the computation starts with wx, $x \in \Sigma^*$, and its result is $|x|$ if it halts. The set of numbers accepted by a P system is the set of results of its computations for all $x \in \Sigma^*$.

2.2 Self-stabilization and related properties

We now resume the discussion started at the end of the Introduction.

Clearly, "a correct state" should be rephrased as "a configuration in the set of correct configurations". Moreover, we would like to eliminate the set of correct states, let us denote it by S, as a parameter. We say that our property holds if there exists some finite set S of configurations satisfying the conditions 1 and 2 above. Since membrane systems are inherently non-deterministic, we additionally propose two weaker degrees of such a property: possible (there exists a computation satisfying the conditions), almost sure (the conditions are satisfied with probability 1 with respect to non-determinism). Finally, if condition 2 is not required, we call the corresponding property (finite) set-convergence instead of self-stabilization. We now give the formal definitions from [1].

Definition 1. *A P system Π is possibly converging to a finite set S of configurations iff for every configuration C of Π there exists a configuration $C' \in S$ such that $C \Rightarrow^* C'$.*

Definition 2. *A P system Π is (almost surely) converging to a finite set S of configurations iff for every configuration C of Π the computations starting in C reach some configuration in S (with probability 1, respectively).*

Definition 3. *A P system Π is possibly closed with respect to a finite set S iff for every non-halting configuration $C \in S$ there exists a configuration $C' \in S$ such that $C \Rightarrow C'$.*

Definition 4. *A P system Π is closed with respect to a finite set S iff for every non-halting configuration $C \in S$ $C \Rightarrow C'$ implies $C' \in S$.*

> We say that a system is **(possibly, almost surely) set-converging** if it is (possibly, almost surely, respectively) converging to some finite set of configurations.
> We say that a system is **possibly self-stabilizing** if it is possibly converging to some finite set S of configurations and if it is possibly closed with respect to S.
> We say that a system is **(almost surely) self-stabilizing** if it is (almost surely, respectively) converging to some finite set S of configurations and if it is closed with respect to S.

The examination of computational aspects of these properties motivates us to add "weakly" to the properties proposed in [1] – *(possibly, almost surely) converging, (possibly) closed, (possibly, almost surely) set-converging, (possibly, almost surely) self-stabilizing* – if the corresponding conditions over configurations C only spans the **reachable non-halting** ones.

Another comment we can make on "almost sure" it that such a property may depend on how exactly the transition probability is defined. The easiest way is to assign equal probabilities to all transitions from a given configuration. Alternatively, to a transition via a multiset of rules $r_1^{n_1} \cdots r_m^{n_m}$ we may assign the weight of a multinomial coefficient $\binom{n_1+\cdots+n_m}{n_1,\cdots,n_m} = \frac{(n_1+\cdots+n_m)!}{n_1!\cdots n_m!}$, which will make the corner cases less probable than the average ones. There can be other ways to define transition probabilities, but we would like to discuss the properties of interest without fixing a specific way. We assume the transition probabilities in an independent subsystem are the same as if it were the entire system.

An important assumption we impose on the probability distribution is that *the probability of each transition is uniquely determined by the associated multiset of rules and by the set of all applicable multisets of rules*, yet it does not depend on the objects that cannot react, or by the previous history of the computation.

2.3 Register machines

In what follows we will need to simulate register machines; here we briefly recall their definition and some of their computational properties. A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, P is the set of instructions bijectively labeled by elements of B, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M can be of the following forms:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  Increase the value of register j by one, and non-deterministically jump to instruction l_2 or l_3. This instruction is usually called *increment*.

- $l_1 : (SUB(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  If the value of register j is zero then jump to instruction l_3, otherwise decrease the value of register j by one and jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stop the execution of the register machine.

A register machine is *deterministic* if $l_2 = l_3$ in all its ADD instructions. A *configuration* of a register machine is described by the contents of each register and by the value of the program counter, which indicates the next instruction to be executed. Computations start by executing the first instruction of P (labeled with l_0), and terminate with reaching a $HALT$-instruction.

Register machines provide a simple universal computational model [3]. Register machines can be used as *accepting* or as *generating* as well as as decision devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts having it as input. Usually, without loss of generality, we may assume that the instruction $l_h : HALT$ always appears exactly once in P, with label l_h. In the generative case, we start with empty registers and take as results of all possible halting computations. Being used as decision devices, register machines may halt in an accepting state with label l_{yes} or in a rejecting state l_{no}, respectively In the following, we shall call a specific model of P systems *computationally complete* if and only if for any register machine M we can effectively construct an equivalent P system Π of that type simulating each step of M in a bounded number of steps and yielding the same results.

3 Results

3.1 Accepting systems

For the following theorem we consider any computationally complete model of P systems as defined above, e.g., a model with maximally parallel multiset rewriting or with controlled sequential multiset rewriting.

Theorem 1. *If a model of P systems yields a computationally complete class, then weakly self-stabilizing subclass accepts exactly $NREC$.*

Proof. For any recursive number set there is a register machine M with one accepting state q_{yes} and one rejecting state q_{no}, deciding it. We modify the register machine in order to obtain a register machine M' which, once the decision is made, i.e., q_{yes} or q_{no} has been reached, erases the workspace and then enters q'_{yes} or q'_{no} respectively, thereby halting in q'_{yes} if and only if the input is accepted or performing an infinite loop with $q'_{no} : (SUB(1), q'_{no}, q'_{no})$ if and only if the input x is rejected. This register machine M' now can be simulated with a P system Π, which by construction starts with a configuration representing the input x and will either end with halting in a configuration representing the state q'_{yes} or else looping in a configuration representing the state q'_{no}, i.e., Π is weakly self-stabilizing.

Conversely, consider a self-stabilizing P system Π, i.e., for each input x, Π performs a computation that ends up in a configuration from a finite set S and then cannot reach any other configuration outside S. Now consider the derivation graph for all possible computations of Π on the input x, i.e., the nodes of this directed graph represent the configurations and the edges indicate the derivation steps from one configuration to the next one during one of these computations. As the number of configurations directly derivable from any configuration in Π is finite, this derivation graph is a connected directed graph with finite degree (from each node, only a finite number of edges is leaving); moreover, this graph cannot have a simple path (a path visiting each node at most once) which is infinite, as every computation in Π has to reach a configuration (node) from S and then cannot leave the set of configurations S any more. Due to König's lemma[4], the total number of nodes (configurations) in the derivation graph must be finite. Hence, even without knowing the set S, the brute force algorithm computing all possible transitions from the initial configuration, but halting as soon as the system halts or a configuration already passed previously is reached, yields a decision procedure for the set accepted by Π. □

Strengthening this result by removing "weakly" is problematic, even if more powerful P systems are used. Indeed, self-stabilization also from unreachable configurations would need to handle not only the configurations without any state or with multiple states (which could be handled with the joint power of maximal parallelism and priorities), but also configurations representing a situation with only one state which is not the initial state of the underlying register machine. We have to leave this question open.

Theorem 2. *If a model of P systems yields a computationally complete class, then the weakly almost surely self-stabilizing P systems of this class accept exactly NRE.*

Proof. We start with the construction from Theorem 1. We want to show that relaxing the property "weakly self-stabilizing" to "almost surely" leads from recursiveness to computational completeness. It suffices to handle the case when the system rejects the input by never halting. We modify the underlying register machine as follows: add a non-deterministic transition from every state $p \in Q$ to a new state e[5], from e erase the contents of all registers and then jump back to e. This will not affect the accepting power, but it will provide a self-stabilizing path from any reachable non-halting configuration.

[4] König's lemma: Let G be a connected graph with finite degree. If G contains an infinite number of nodes, then it contains an infinite simple path.

[5] The transition from p to e can be done by $p : (ADD(j), e, e)$, since the registers then are emptied anyway. Furthermore, the basic model of register machines does not allow non-determinism other than $p : (ADD(j), q, r)$. The branching at ADD instructions might be done by assuming the original computation to be deterministic and replacing $p : (ADD(j), q, q)$ by $p : (ADD(j), q, e)$. The branching at a SUB instruction $p : (SUB(j), q, r)$ may be done by the sequence of rules $p : (ADD(j), e, p')$, $p' : (SUB(j), p'', p'')$, $p'' : (SUB(j), q, r)$.

The probability that the computation does not self-stabilize for more than k steps decreases exponentially with respect to k. Indeed, the simulation of a register machine by P system has bounded parallelism, each instruction is simulated in a bounded number of steps, and at least one path leads to self-stabilization. Moreover, there only exists a finite number of different sets of applicable multisets containing a branching from the simulation into the self-stabilization path, so the minimum probability for this self-stabilization path is strictly positive. These observations conclude the proof. □

Theorem 3. *If a model of P systems yields a computationally complete class, then the class of all almost surely self-stabilizing maximally parallel/sequential P systems with priorities accepts exactly NRE.*

Proof. Given a set L from NRE, we first construct a P system Π simulating a register machine M accepting L and then extend Π to a P system Π' even fulfilling the condition of being almost surely self-stabilizing.

Let $M = (m, B, l_0, l_h, P)$ a deterministic register machine accepting L. We now construct the P system $\Pi = (O, l_0, R, >)$ with priorities accepting L:

$$\begin{aligned}
O &= B \cup \{a_i \mid 1 \leq i \leq m\}\,, \\
R &= \{l_1 \to a_j l_2 \mid l_1 : (ADD\,(j)\,, l_2) \in P\} \\
&\quad \cup \{a_j l_1 \to l_2, l_1 \to l_3 \mid l_1 : (SUB\,(j)\,, l_2, l_3) \in P\} \\
> &= \{a_j l_1 \to l_2 > l_1 \to l_3 \mid l_1 : (SUB\,(j)\,, l_2, l_3) \in P\}\,.
\end{aligned}$$

The contents of a register i, $1 \leq i \leq m$, is represented by the number of symbols a_i in Π. The state l of the register machine is represented by the corresponding symbol l in Π, too. When M halts in l_h with all registers being empty, Π also halts with the configuration $\{l_h\}$. Obviously, Π accepts L, both in the sequential as well as in the maximally parallel mode.

To strengthen the result to even non-weak almost sure self-stabilization, we have to take into account the non-reachable configurations, too. The almost surely self-stabilizing P system $\Pi' = (O', l_0, R', >')$ with priorities accepting L is constructed as follows:

$$\begin{aligned}
O' &= B \cup \{a_i \mid 1 \leq i \leq m\} \cup \{e\}\,, \\
R' &= \{l_1 \to a_j l_2 \mid l_1 : (ADD\,(j)\,, l_2) \in P\} \\
&\quad \cup \{a_j l_1 \to l_2, l_1 \to l_3 \mid l_1 : (SUB\,(j)\,, l_2, l_3) \in P\} \\
&\quad \cup \{a_i \to e \mid 1 \leq i \leq m\} \cup \{ex \to e \mid x \in O'\} \cup \{e \to e\} \\
&\quad \cup \{l \to e \mid l \in B \setminus \{l_h\}\} \cup \{ll' \to e \mid l, l' \in B\}\,, \\
>' &= \{a_j l_1 \to l_2 > l_1 \to l_3 \mid l_1 : (SUB\,(j)\,, l_2, l_3) \in P\} \\
&\quad \cup \{ex \to e > r, ll' \to e > r \mid l, l' \in B, x \in O', r \in R\} \\
&\quad \cup \{l \to e > a_i \to e \mid l \in B \setminus \{l_h\}\,, 1 \leq i \leq m\} \\
&\quad \cup \{r > e \to e \mid r \in R' \setminus \{e \to e\}\}\,.
\end{aligned}$$

In addition to the idea of the construction given in the proof of Theorem 2 using the exit e by applying a rule $l \to e$, $l \in B \setminus \{l_h\}$, it suffices to self-stabilize from the configurations with no state and from the configurations with multiple states of the register machine. Multiple states can be reduced by the rules $ll' \to e$, $l, l' \in B$. If no state symbol is present, then we may exit with one of the rules $a_i \to e$, $1 \leq i \leq m$. All remaining cases can be captured by the rules $ex \to e$, $x \in O'$. By construction, the self-stabilizing set S equals $\{\{l_h\}, \{e\}\}$. The whole construction again is valid for the sequential as well as the maximally parallel mode. □

An open question is whether priorities in Theorem 3 can be replaced by promoters or inhibitors.

3.2 Generating systems

Theorem 4. *Any finite set M of numbers can be generated by some self-stabilizing membrane system without control.*

Proof. Consider a P system $\Pi = (\{s, a\}, s, R)$, where

$$R = \{s \to a^n \mid n \in M\} \cup \{a^{max(M)+1} \to \lambda, ss \to s\}.$$

It is not difficult to see that Π generates M and (taking $S = \{a^n \mid n \leq max(M)\} \cup \{s\}$) it is self-stabilizing. □

Since self-stabilization implies set-convergence and closure, and relaxing either property (to possibly, almost surely and/or weakly) does not compromise the construction of the P system descibed in the proof of Theorem 4, the lower bound on the generative power of associated systems restricted to any property we have defined, is at least $NFIN$.

Lemma 1. *A possibly finite set-converging system only generates finite sets.*

Proof. It follows from Definition 1 that for a system possibly converging to a set S, S contains all halting configurations. Since S is finite, so is the set of all the halting configurations. Hence, at most $NFIN$ can be generated. □

Theorem 5. *Any of the following classes of P systems* $\mathbf{dp}OP^{\mathbf{m}}(\mathbf{c})$ *generate exactly $NFIN$:*

- **d** *is possibly/almost surely/ -*
- **p** *is self-stabilizing/finite set-converging*
- **m** *is maximally parallel/sequential*
- **c** *is uncontrolled/with promoters/with inhibitors/with priorities.*

Proof. The claims of the theorem directly follow from Theorems 4 and 5.

We now proceed to weak properties of generative systems.

Theorem 6. *Weakly almost surely self-stabilizing P systems generate exactly $NFIN$.*

Proof. The lower bound is shown by Theorem 4. Now take a weakly self-stabilizing P system Π, and its associated set S from the definition of the property. Consider an arbitrary halting computation of Π. Let C be the configuration of Π one step before the halting. Interpreting finite set-convergence for C implies that the halting configuration belongs to S. Since the halting computation has been arbitrarily chosen, the set of all halting configurations is a subset of S, and hence it is finite. Therefore, the set generated by Π is finite, too. □

Theorem 7. *If a model of P system yields a computationally complete class, then weakly possibly self-stabilizing subclass generates* NRE*.*

Proof. Consider the construction from Theorem 2, but for a generative P system. The simulation of the underlying register machine is carried out until some point. Unless the P system has already halted, it always has a choice to self-stabilize and loop. □

4 Conclusions

We have presented some results concerning the concept of self-stabilization, recently proposed for membrane computing. Its essence is in reachability and closure of a finite set.

Some of the obtained results can be summarized in the following table:

Property	computationally complete	(sequ/maxpar)+pri	Thm
self stabilizing	acc. ?/gen. $NFIN$		-/5
almost surely s.s.	acc. ?/gen. $NFIN$	acc. NRE/gen. $NFIN$	3/5
possibly s.s.	acc. ?/gen. $NFIN$	acc. NRE/gen. $NFIN$	3/5
weakly s.s.	acc. $NREC$/gen. $NFIN$		1/6
weakly almost surely s.s.	acc. NRE/gen. $NFIN$		2/6
weakly possibly s.s.	acc. NRE/gen. NRE		2/7

One of the questions we proposed is whether priorities may be replaced by promoters or inhibitors in Theorem 2. Another open question is the power of accepting with unrestricted self-stabilization, even if maximal parallelism is combined with priorities (a comment after Theorem 1 and the first question mark in the table above). The other open questions are also marked with question marks in the table above. Any system in the corresponding classes must (besides doing the actual computation) converge (definitely, in probability or possibly) to some finite set from anywhere, without using the joint power of maximal parallelism and control.

We mention two topics that we do not deal with here. One is considering the finite set as a parameter, possibly leading to a discussion in model checking. The other one concerns reachability questions in dynamic membrane structures.

Acknowledgements

The first author gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618). The work of the second, the fourth and the fifth author was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

References

1. A. Alhazov: Properties of Membrane Systems. *Membrane Computing*, 12th International Conference, CMC 2011, Fontainebleau, Revised Selected Papers (M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, S. Verlan, Eds.), *Lecture Notes in Computer Science* **7184**, 2012, 1–13.
2. E.A. Emerson: Temporal and Modal Logic. In: *Handbook of Theoretical Computer Science*, Chapter 16, the MIT Press, 1990.
3. M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
4. Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
5. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
6. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
7. P systems webpage: `http://ppage.psystems.eu`
8. `http://en.wikipedia.org/wiki/Self-stabilization`

Characterizing the Computational Power of Energy-Based P Systems

Artiom Alhazov[1,2], Marco Antoniotti[1], Alberto Leporati[1]

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
{artiom.alhazov,marco.antoniotti,alberto.leporati}@unimib.it

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

Summary. We investigate the computational power of energy-based P systems, a model of membrane systems where a fixed amount of energy is associated with each object and the rules transform single objects by adding or removing energy from them. We answer recently proposed open questions about the power of such systems without priorities associated to the rules, for both sequential and maximally parallel modes. We also conjecture that deterministic energy-based P systems are not computationally complete.

1 Introduction

Membrane systems (also called P systems) have been introduced in [11] as a class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. Since then, many variants of P systems have been defined in the literature. In what follows we assume the reader is familiar with the basic notions and the terminology underlying P systems. A systematic introduction to the area can be found in [12]; a recent overview of the developments is presented in [13], whereas the latest information can be found in [15].

In this paper we consider *energy-based P systems* [7, 8, 6], a model of computation in the framework of Membrane Computing in which a given amount of energy is associated to each object, and the energy manipulated during computations is taken into account by means of conservative rules.

Let us note in passing that there has been other attempts in the literature to incorporate certain conservation laws in membrane computing. One is purely communicative models, of which the most thoroughly studied is P systems with symport/antiport [10]. In these systems the computation is carried out by moving objects between the regions in groups. To reach computational completeness, the workspace is increased by bringing (some types of) objects from the environment, where they can be found in an unbounded supply. Another model is *conformon P systems* [5], where computations are performed by redistributing energy between objects, than can also be renamed and moved. A feature of these systems is that a different amount of energy may be embedded in the same object at different time steps. Yet another approach is to assign energy to membranes, as in P system with Unit Rules and Energy assigned to Membranes (*UREM P systems*, for short) [1]. Here the computations are performed by rules renaming and moving an object across a membrane, possibly modifying the energy assigned to that membrane. It has been proved in [1] that UREM P systems working in the sequential mode characterize $PsMAT$, the family of Parikh sets generated by matrix grammars without appearance checking (and with *erasing rules*), and that their power is increased to $PsRE$ (the family of recursively enumerable Parikh sets) if priorities are assigned to the rules or the mode of applying the rules is changed to maximally parallel.

As stated above, in this paper we consider *energy-based P systems*, in which energy is assigned to objects in a way that each object from the alphabet is assigned a specific value. Instances of a special symbol are used to denote *free energy units* occurring inside the regions of the system. The computations are carried out by rules renaming and possibly moving objects, which may consume or release free energy in the region, respecting the energy conservation law (that is, the total amount of energy associated with the objects that appear in the left hand side of a rule is the same as the energy occurring in the right hand side). The result of a computation may be interpreted in many ways: for example, as the amount of free energy units in a designated output region. Also for this model, to give the possibility to reach computational completeness it is necessary (but not sufficient, as we will see) that there may be an unbounded amount of free energy in (at least one) specified region of the system. In [6] it is proved that energy-based P systems working in the sequential way and using a form of local priorities associated to the rules are computationally complete. Without priorities, their behavior can be simulated by vector addition systems, and hence are not universal. However, in [6] the precise characterization of the computational power of energy-based P systems without priorities is left as an open problem. A related open question was whether energy-based P systems can reach computational completeness by working in the maximally parallel mode, without priorities, as it happens with UREM P systems [1].

In this paper we answer these questions, by showing that the power of energy-based P systems containing an infinite amount of free energy and without priorities is exactly $PsMAT$ when working in the sequential mode, and $PsRE$ when working in the maximally parallel mode. Nonetheless we will end with another open question: what is the power of energy-based P systems under the restriction of determinism? We conjecture non-universality for this case.

The rest of the paper is structured as follows. The next section contains some mathematical preliminaries, to fix the notions, definitions and notations with which we will work. Section 3 contains our results concerning the characterization of the computational power of energy-based P systems working without priorities, either in the sequential or in the maximally parallel mode. Section 4 contains the conclusions and some discussion on the above mentioned open problem, concerning the computational power of deterministic energy-based P systems.

2 Preliminaries

We assume the reader to be familiar with the basics of formal languages; on this subject one may refer to, e.g., [14].

We denote by $\mathbb{N}$ the set of non-negative integers. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. By $\mid x \mid$ we denote the length of the word x over V. Let $\{a_1, \ldots, a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $\mid x \mid_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, \ldots, a_n$ is $(\mid x \mid_{a_1}, \ldots, \mid x \mid_{a_n})$. The *Parikh image* of a language L over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in L. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. A finite multiset $\langle m_1, a_1\rangle \ldots \langle m_n, a_n\rangle$ with $m_i \in \mathbb{N}$, $1 \leq i \leq n$, is represented as any string x the Parikh vector of which with respect to $a_1, \ldots, a_n$ is $(m_1, \ldots, m_n)$. The family of recursively enumerable languages is denoted by RE, and the family of context-free languages by CF. The family of all recursively enumerable sets of k-dimensional vectors of non-negative integers can thus be denoted by $Ps(k)RE$. Since numbers can be seen as one-dimensional vectors, we can replace $Ps(1)$ by $\mathbb{N}$ in the notation, thus obtaining $\mathbb{N}RE$.

2.1 Matrix Grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = [m_{i,1}, \ldots, m_{i,n_i}]$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over (N, T). For $m_i = [m_{i,1}, \ldots, m_{i,n_i}]$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$\begin{aligned} L(G) = \{w \in T^* \mid\ & S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w, \\ & w_j \in (N \cup T)^*,\ m_{i_j} \in M \ \text{ for } 1 \leq j \leq k, k \geq 1\}. \end{aligned}$$

According to the definitions given in [2], the last matrix can already finish with a terminal word without having applied the whole sequence of productions. The family of languages generated by matrix grammars without appearance checking is denoted by MAT. It is known that $PsCF \subset PsMAT \subset PsRE$. Further details about matrix grammars can be found in [2] and in [14].

2.2 Energy-based P systems

Let us now recall the definition of energy-based P systems as given in [8].

An *energy-based P system* of degree $m \geq 1$ is a tuple

$$\Pi = (A, \varepsilon, \mu, e, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{in}, i_{out})$$

where:

- A is a finite set of objects called the alphabet;
- $\varepsilon : A \to \mathbb{N}$ is a mapping that associates to each object $a \in A$ the value $\varepsilon(a)$ (also denoted by ε_a), which can be viewed as the "energy value of a". If $\varepsilon(a) = \ell$, we also say that object a *embeds* ℓ units of energy;
- μ is a description of a tree structure consisting of m membranes, injectively labeled with elements from the set $\{1, \ldots, m\}$;
- $e \notin A$ is a special symbol denoting one free energy unit;
- w_i, for $1 \leq i \leq m$, specifies multisets (over $A \cup \{e\}$) of objects initially present in region i. We will sometimes assume that the number of e's (but not of objects from A) in some regions of the system is unbounded;
- R_i, for $1 \leq i \leq m$, is a finite set of multiset rewriting rules over $A \cup \{e\}$ associated with region i. Rules can be of the following types:

  $$ae^k \to (b, p) \qquad \text{and} \qquad b \to (a, p)e^k \tag{1}$$

  where $a, b \in A$, $p \in \{here, in(j), out \mid 1 \leq j \leq m\}$, and k is a non-negative integer. Rules satisfy the conservativeness condition $\varepsilon(a) + k = \varepsilon(b)$;
- $i_{in} \in \{1, 2, \ldots, m\}$ specifies the input region of Π;
- $i_{out} \in \{0, 1, \ldots, m\}$ specifies the output region of Π ($i_{out} = 0$ corresponds to the environment).

Remark 1. In the above definition we excluded rules of types $e \to (e, p)$, originally included in [8]. It is easy to see that this does not influence the computational power of energy-based P systems.

When a rule of the type $ae^k \to (b, p)$ is applied, the object a, in presence of k free energy units, is allowed to be transformed into object b (note that $\varepsilon_a + k = \varepsilon_b$, for the conservativeness condition). If $p =$ here, then the new object b remains in the same region; if $p =$ out, then b exits from the current membrane. Finally, if $p =$ in($name$), then b enters into the membrane labelled with $name$, which must be directly contained inside the current membrane in μ. The meaning of rule $b \to (a, p)e^k$, where k is a positive integer number, is similar: the object b is allowed to be transformed into object a by releasing k units of free energy (also here, $\varepsilon_b = \varepsilon_a + k$). As above, the new object a may optionally move one level up or down into the membrane structure. The k free energy units might then be used by another rule to produce "more energetic" objects from "less energetic" ones. When $k = 0$ the rule $ae^k \to (b, p)$, also written as $a \to (b, p)$, transforms the object a into the object b (note that in this case $\varepsilon_b = \varepsilon_a$) and moves it (if $p \neq$ here) upward or downward into the membrane hierarchy, without acquiring or releasing any free energy unit. A similar observation applies to rules $b \to (a, p)e^k$ when $k = 0$.

Rules can be applied either in the sequential or in the maximally parallel mode. In either cases, we assume that the execution of rules does not consume energy. When working in the *sequential mode*, at each computation step (a global clock is assumed) exactly one enabled rule is nondeterministically chosen and applied in the whole system. When working in the *maximally parallel mode*, instead, at each computation step in each region of the system a maximal multiset of applicable rules is selected, and then all those rules are applied in parallel. Here *maximal* means that no further rule is applicable to objects that are "idle", that is, not already used by some other rule. If two or more maximal sets of applicable rules exist, then one of them is nondeterministically chosen.

A configuration of Π is the tuple $(M_1, \ldots, M_m)$ of multisets (over $A \cup \{e\}$) of objects contained in each region of the system; $(w_1, \ldots, w_m)$ is the *initial* configuration. A configuration where no rule can be further applied on is said to be *final.* A computation is a sequence of transitions between configurations of Π, starting from the initial one. A computation is *successful* if and only if it reaches a final configuration or, in other words, it *halts.* The multiset $w_{i_{\text{in}}}$ of objects occurring inside the input membrane is the *input* for the computation, whereas the multiset of objects occurring inside the output membrane (or ejected from the skin, if $i_{\text{out}} = 0$) in the final configuration is the *output* of the computation. A non-halting computation produces no output. As an alternative, we can consider the Parikh vectors associated with the multisets, and see energy-based P systems as computing devices that transform (input) Parikh vectors to (output) Parikh vectors. We may also assume that energy-based P systems have $\alpha \geq 1$ input membranes and $\beta \geq 1$ output membranes, instead of one. This modification does not increase the computational power of energy-based P systems, since for any fixed value of $\alpha \geq 1$ (resp., $\beta \geq 1$), the set $\mathbb{N}^\alpha$ (resp., $\mathbb{N}^\beta$) is isomorphic to $\mathbb{N}$, as it is easily shown by using the well known Cantor mapping.

In what follows sometimes we will use energy-based P systems as generating devices: we will disregard the input membrane, and will consider the multisets (or Parikh vectors) produced in the output membrane at the end of the (halting) nondeterministic computations of the system. In particular, in the output multisets we will only count the number of free energy units contained in the β output regions in the final configuration. We will denote the family of β-dimensional vectors generated in this way by energy-based P systems with at most m membranes and unbounded energy by $Ps(\beta)OP_m(energy_*)$. The union of all these classes for β ranging through the set of all non-negative integers is denoted by $PsOP_m(energy_*)$. When $\beta = 1$, the class $Ps(\beta)OP_m(energy_*)$ will be written as

$\mathbb{N}OP_m(energy_*)$. In all cases we will replace the subscript m by $*$ if no bound is placed on the number of membranes. If instead of maximal parallelism we assume that the P system evolves sequentially, we will add the superscript *seq* to P in the notation.

Our results will thus be proved for energy-based P systems working as generating devices; however, the extension to the cases in which P systems are computing functions or are accepting multisets of objects (or Parikh vectors) will be straightforward.

2.3 Register machines

In what follows we will need to simulate register machines; here we briefly recall their definition and some of their computational properties. A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, P is the set of instructions bijectively labeled by elements of B, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M can be of the following forms:

- $l_1 : ADD(j, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  Increase the value of register j by one, and nondeterministically jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : SUB(j, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  If the value of register j is zero then jump to instruction l_3, otherwise decrease the value of register j by one and jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stop the execution of the register machine. Note that, without loss of generality, we may assume that this instruction always appears exactly once in P, with label l_h.

A register machine is *deterministic* if $l_2 = l_3$ in all its ADD instructions. A *configuration* of a register machine is described by the contents of each register and by the value of the program counter, that indicates the next instruction to be executed. Computations start by executing the first instruction of P (labelled with l_0), and terminate if and when they reach instruction l_h.

Register machines provide a simple universal computational model [9]. In particular, the results proved in [4] immediately lead to the following proposition.

Proposition 1. *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\}+2)$-register machine M computing f in such a way that, when starting with $(n_1, \ldots, n_\alpha) \in \mathbb{N}^\alpha$ in registers 1 to α, and all other registers empty, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label l_h with registers 1 to β containing r_1 to r_β, and all other registers being empty. If the final label cannot be reached, then $f(n_1, \ldots, n_\alpha)$ remains undefined.*

Register machines can also be used as *accepting* or as *generating* devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts. The following Proposition is a direct consequence of Proposition 1.

Proposition 2. *For any recursively enumerable set $L \subseteq Ps\,(\alpha)\,RE$ of vectors of non-negative integers there exists a deterministic register machine M with $(\alpha + 2)$ registers accepting L in such a way that, when starting with n_1 to n_α in registers 1 to α, M has accepted $(n_1, \ldots, n_\alpha) \in L$ if and only if it halts in the final label l_h with all registers being empty.*

To generate vectors of non-negative integers we have to use nondeterministic register machines. The following Proposition is also a direct consequence of Proposition 1.

Proposition 3. *For any recursively enumerable set $L \subseteq Ps\,(\beta)\,RE$ of vectors of non-negative integers there exists a nondeterministic register machine M with $(\beta + 2)$ registers generating L in such a way that, when starting with all registers being empty, M has generated $(r_1, \ldots, r_\beta) \in L$ if it halts in the final label l_h with registers 1 to β containing r_1 to r_β, and all other registers being empty.*

A direct consequence of the results exposed in [9] is that in Propositions 1 and 3 we may assume without loss of generality that only ADD instructions are applied to the output registers. This fact will be used to decrease the number of membranes of energy-based P systems simulating register machines; in particular, for each output register one membrane will suffice, whereas to simulate the behaviour of the other registers we will need two membranes.

A register machine is called *partially blind* if performing a zero-test blocks the computation, thus leading to no result. We can reflect this situation by omitting l_3 from all SUB instructions. However, unless all non-output registers have value zero at halting, the result of a computation is discarded; note that this is an implicit final zero-test, imposed by the definition and not affecting the power of register machines in the general case. It is known [3] that partially blind register machines characterize $PsMAT$.

3 Characterizing the Power of Energy-based P Systems

In this section we characterize the computational power of energy-based P systems without priorities associated to the rules and with an unbounded amount of free energy units. As stated in the previous section, we use energy-based P systems as generating devices; the extension to the computing and accepting cases (concerning computational completeness) is easy to obtain.

We start with systems working in the sequential mode; with the next two theorems we prove that they characterize $PsMAT$. We first prove the inclusion $PsMAT \subseteq PsOP_*^{seq}(energy_*)$, obtained by simulating partially blind register machines.

Theorem 1. *$PsOP_*^{seq}(energy_*) \supseteq PsMAT$.*

Proof. Consider a partially blind register machine $M = (m, B, l_0, l_h, P)$, generating β-dimensional vectors, and assume that registers $\beta+1, \ldots, m$ are empty in the final configuration of successful computations, corresponding to an implicit final zero-test.

We construct a corresponding energy-based P system Π_M, containing an infinite amount of free energy units in the skin, as follows:

$$\begin{aligned}
\Pi &= (A, \varepsilon, \mu, e, w_s, \ldots, w_f, R_s, \ldots, R_f), \text{ where}\\
A &= \{l, l', l'' \mid l \in B\} \cup \{t_j, T_j \mid \beta+1 \leq j \leq m\} \cup \{t, T, H, H'\},\\
\varepsilon(l) &= 1,\ \varepsilon(l') = 2,\ \varepsilon(l'') = 0,\ l \in B,\\
\varepsilon(t_j) &= 0,\ \varepsilon(T_j) = 2,\ \beta+1 \leq j \leq m,\\
\varepsilon(t) &= 0,\ \varepsilon(T) = 1,\ \varepsilon(H) = 2m+1,\ \varepsilon(H') = 2\beta+2,\\
\mu &= [\ [\ ]_1 \cdots [\ ]_m\ [\ ]_f\ ]_s\,,\\
w_s &= l_0,\\
w_j &= \lambda,\ 1 \leq j \leq m,\\
w_f &= t_{\beta+1} \cdots t_m T,\\
R_s &= \{l_1 e \rightarrow (l_1', in(j)) \mid l_1 : ADD(j, l_2, l_3) \in P\}\\
&\quad \cup \{l_1 \rightarrow (l_1'', in(j))\, e \mid l_1 : SUB(j, l_2) \in P\}\\
&\quad \cup \{T \rightarrow (T, in(f)),\ l_h e^{2m} \rightarrow (H, in(f))\}\\
&\quad \cup \{T_j \rightarrow (t, in(j))\, e^2 \mid \beta+1 \leq j \leq m\}.\\
R_j &= \{l_1' \rightarrow (l_2, out)\, e,\ l_1' \rightarrow (l_3, out)\, e \mid l_1' : ADD(j, l_2, l_3) \in P\}\\
&\quad \cup \{l_1'' e \rightarrow (l_2, out) \mid l_1 : SUB(j, l_2) \in P\} \cup R_j',\ 1 \leq j \leq m,\\
R_j' &= \emptyset,\ 1 \leq j \leq \beta,\\
R_j' &= \{T \rightarrow te,\ te \rightarrow T\},\ \beta+1 \leq j \leq m,\\
R_f &= \{T \rightarrow te,\ te \rightarrow T,\ H \rightarrow H' e^{2(m-\beta)-1}\}\\
&\quad \cup \{t_j e^2 \rightarrow (T_j, out) \mid \beta+1 \leq j \leq m\}.
\end{aligned}$$

Note that if $\beta = m$, then we replace $H \rightarrow H'e^{-1}$ in R_f by $He \rightarrow H'$. The sets of rules R_j and R_j', $1 \leq j \leq m$, are both intended to be associated with region j, and hence should be joined. As explained below, the rules from R_j' are used to produce infinite $T \leftrightarrow te$ loops in the regions corresponding to non-output registers of M if such registers are nonempty when the computation halts.

The simulation consists of a few parts. Every object associated with an instruction label l_1 embeds 1 unit of energy. To simulate an increment instruction $l_1 : ADD(j, l_2, l_3)$, the corresponding object l_1 consumes 1 more unit of energy, enters the region associated with register j as l_1', releases e there, and returns to the skin either as the object l_2 or as l_3 (the choice being made in a nondeterministic way), indicating the next instruction of M to be simulated. To simulate a decrement instruction $l_1 : SUB(j, l_2)$, the corresponding object l_1 releases e in the skin and enters as l_1'' the region associated with register j. There it consumes 1 unit of energy, and returns to the skin as the object l_2 associated with the next instruction of M to be simulated. If the register was empty then this process is blocked.

Meanwhile, another process takes place in region f; the order of execution of the two processes is nondeterministic, but both must finish in order for the system to terminate the computation and produce a result. The aim of this latter process is indeed to make Π_M "compute" forever if the above simulation of the SUB instruction gets blocked when trying to decrement an empty register, so that no result is produced in this case. The process consists of object T cyclically releasing e and capturing it, generating a possibly infinite loop. The only way to stop the loop is to alter the free energy occurring in region m. This is done when the simulation of M is finished, leading to object H releasing $e^{2(m-\beta)-1}$. If $\beta = m$ this consumes 1 unit of energy, thus stopping the $T \leftrightarrow te$ loop. Otherwise it releases enough energy for objects t_j, $\beta + 1 \leq j \leq m$ to leave the region, and the last one of them stops the $T \leftrightarrow te$ loop. The objects t_j are used to ensure that registers $\beta + 1 \leq j \leq m$ are empty, otherwise causing a $T \leftrightarrow te$ loop in the corresponding region. □

The opposite inclusion, $PsOP_*^{seq}(energy_*) \subseteq PsMAT$, is proved by simulating energy-based P systems by matrix grammars.

Theorem 2. *$PsOP_*^{seq}(energy_*) \subseteq PsMAT$.*

Proof. Let Π be an energy-based P system containing an infinite amount of free energy units and applying the rules in the sequential mode. Each rule of Π can be simulated by a corresponding rewriting rule on multisets of object-region pairs, ignoring those pairs involving energy objects in the regions containing infinite free energy. Such a multiset rewriting rule can be written as a matrix, yielding a matrix grammar. Clearly no matrix can be applied if and only if no rule can be applied. Since matrix languages are closed under morphisms, when this happens we can apply a morphism that erases all object-region pairs except those involving free energy objects in the output regions. The resulting language belongs to MAT, and its Parikh mapping yields exactly $Ps(\Pi)$. □

By joining the two inclusions proved in Theorems 1 and 2 we obtain our characterization of $PsMAT$ by energy-based P systems working in the sequential mode with an unbounded amount of free energy:

Corollary 1. *$PsOP_*^{seq}(energy_*) = PsMAT$.*

Running energy-based P systems in the maximally parallel mode allows them to reach computational completeness without using priorities, as shown in the next theorem.

Theorem 3. *$Ps(\beta)RE = Ps(\beta)OP_{\beta+6}(energy_*)$ for all integers $\beta \geq 1$.*

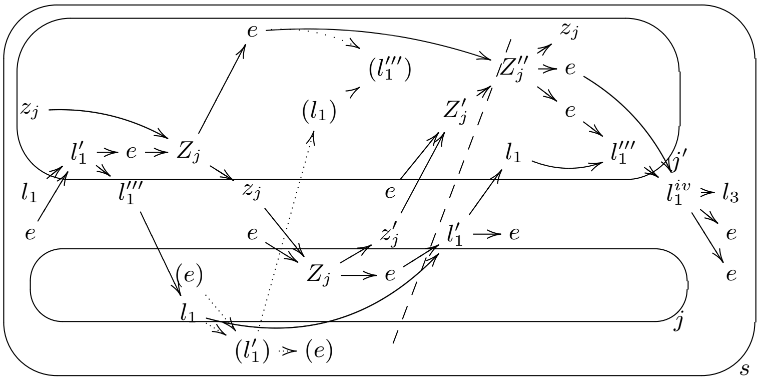


Fig. 1. Simulation of the zero-test of $l_1 : SUB(j, l_2, l_3)$. The case when the register is not zero is shown by dotted lines and symbols in parentheses, and the computation stops before the dashed line.

Proof. We start by noticing that the construction from Theorem 1 produces the same result when the P system works in the maximally parallel mode. Thus, it suffices to only add a simulation of zero-test instructions without disrupting the existing machinery. The new system nondeterministically chooses between decrement and zero-test, and blocks the simulation process if the zero-test fails.

We thus add membranes $[\ ]_{1'}$, $[\ ]_{2'}, \dots, [\ ]_{m'}$ and the following sets of rules to the energy-based P system Π_M mentioned in the proof of Theorem 1:

$$\begin{aligned}
R_s^0 = \{ & 1 : l_1 e \to (l_1', in(j')),\ 3a : l_1''' \to (l_1, in(j)), \\
& 5a : l_1' \to (l_1, in(j'))\, e,\ 5b : z_j e \to (Z_j, in(j))\, e, \\
& 7b : z_j' e \to (Z_j', in(j')),\ 12a : l_1^{(iv)} \to l_3 e^2 \\
& \mid l_1 : SUB(j, l_2, l_3) \in P\}, \\
R_j^0 = \{ & 4a : l_1 e \to (l_1', out),\ 6b : Z_j \to (z_j', out)\, e \in R_j'' \\
& \mid l_1 : SUB(j, l_2, l_3) \in P\}, \\
R_{j'}^0 = \{ & 2 : l_1' \to (l_1''', out)\, e,\ 3b : z_j e \to Z_j,\ 4b : Z_j \to (z_j, out), \\
& 8b : Z_j' e \to Z_j'',\ 9b : Z_j'' \to z_j e^2,\ 10a : l_1 e \to l_1''', \\
& 11a : l_1''' e \to (l_1^{(iv)}, out) \mid l_1 : SUB(j, l_2, l_3) \in P\}.
\end{aligned}$$

The case of correct simulation of a zero-test is illustrated in Figure 1. Indeed, if region j does not contain any object e, then the following sequence of multisets of rules is applied: $1, 2, (3a, 3b), (4a, 4b), (5a, 5b)$,$6b, 7b, 8b, 9b$, $10a, 11a$, $12a$. In this way, l_1 is transformed to l_3 while the other objects used (energy and z_j) are reproduced. On the other hand, if region j contains some object e, corresponding to a non-zero value of the corresponding register, then the sequence of multisets of rules applied is $1, 2, (3a, 3b), (4a, 4b), (5a, 5b), (6b, 10a), 7b$, and the simulation process is blocked. We recall that blocking the simulation process leads to an infinite computation due to the $T \leftrightarrow te$ loop in region f. In words, because of free energy in region j, object l_1 left that region three steps earlier. As a result, instead of object Z'_j consuming 1 unit of energy and then releasing 2 units needed for l_1, the existing unit of energy has been consumed by l_1, leaving the computation unfinished.

The full P system, defined using components of the construction from Theorem 1, is given below:

$$\begin{aligned}
\Pi'' &= (A'', \varepsilon'', \mu'', e, w''_s, \ldots, w''_f, R''_s, \ldots, R''_f), \text{ where}\\
A'' &= A \cup \{l''' \mid l \in B\} \cup \{z_j, z'_j, Z_j, Z'_j, Z''_j \mid 1 \le j \le m\},\\
\varepsilon''(x) &= \varepsilon(x),\ \forall\, x \in A,\ \varepsilon(l''') = 1,\ \varepsilon(l^{(iv)}) = 2,\ \forall\, l \in B,\\
\varepsilon''(z_j) &= \varepsilon''(z'_j) = 0,\ \varepsilon''(Z_j) = \varepsilon''(Z'_j) = 1,\ \varepsilon''(Z''_j) = 2,\ 1 \le j \le m,\\
\mu &= \ [\ [\ ]_1\ \cdots\ [\ ]_m\ [\ ]_{1'}\ \cdots\ [\ ]_{m'}\ [\ ]_f\ ]_s\ ,\\
w''_s &= w_s,\ w''_j = w_j,\ 1 \le j \le m,\ w''_f = w_f,\\
w''_{j'} &= z_j,\ 1 \le j \le m,\\
R''_s &= R_s \cup R^0_s,\ R''_j = R_j \cup R^0_j,\ 1 \le j \le m,\\
R''_{j'} &= R^0_{j'},\ 1 \le j \le m,\ R''_f = R_f.
\end{aligned}$$

As we can see, system Π'' uses the skin membrane, one membrane to control the halting, and two membranes for each of the m registers, for a total of $2m + 2$ membranes.

Since it is known (see Proposition 3) that $m = \beta + 2$ registers suffice to generate any recursively enumerable set $L \subseteq Ps(\beta)RE$ of vectors of non-negative integers by nondeterministic register machines, we would obtain $2\beta + 6$ membranes. However, as recalled above, when using register machines as generating devices we can assume without loss of generality that only ADD instructions are applied to the output registers. So the number of membranes needed to simulate M reduces to $\beta + 2(m - \beta) + 2 = \beta + 6$. □

By putting $\beta = 1$ in the above theorem we obtain a characterization of $\mathbb{N}RE$:

Corollary 2. *$\mathbb{N}OP_7(energy_*) = \mathbb{N}RE$.*

whereas if we make the union of all classes $Ps(\beta)OP_{\beta+6}(energy_*)$ for β ranging through the set of non-negative integers we obtain a characterization of $PsRE$:

Corollary 3. *$PsOP_*(energy_*) = PsRE$.*

As stated above, these results can be easily generalized to the cases in which energy-based P systems are used as accepting devices or as devices computing partial recursive functions. First of all note that the energy-based P systems built in the proofs of Theorems 1 and 3 can be easily modified to simulate deterministic register machines. Considering the computing case, we know from Proposition 1 that $m = \max\{\alpha, \beta\} + 2$ registers suffice to compute any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$. To simulate such a register machine we would obtain $2\max\{\alpha, \beta\} + 6$

membranes for the system Π'' built in the proof of Theorem 3. However, this number can be reduced to $\alpha + \max\{\alpha, \beta\} + 6$ by considering that:

- as stated above, we can assume that only *ADD* instructions are applied to the output registers. This means that only one membrane (instead of two) is needed to simulate the behaviour of each output register;
- in general some input registers may also be used as output registers. However, any "primed" membrane j' associated with an input register, $1 \leq j' \leq \alpha$, cannot be used also as a membrane associated to an output register, due to the object z_j residing in the membrane. Hence, with α inputs and β outputs we need α primed membranes plus $\max\{\alpha, \beta\}$ non-primed membranes. By adding two membranes for each of the 2 additional registers of M, plus membranes f and s, we obtain $\alpha + \max\{\alpha, \beta\} + 6$ membranes.

As particular cases, we need $2\alpha + 6$ membranes for the accepting case and $\beta + 6$ membranes for the generating case.

4 Conclusions and Future Work

In this paper we have considered *energy-based P systems*, a model of membrane systems with energy assigned to objects. We have answered two questions about their computational power, and we have thus proved that it matches Parikh mapping of matrix languages when the rules of the P systems are applied in the sequential mode, whereas there is computational completeness in the maximally parallel case.

As a direction for future research, we propose the following problem: What is the computational power of *deterministic* energy-based P systems? We conjecture that they are not universal. The question originates from the fact that in [7, 8] energy-based P systems are used to simulate Fredkin gates and Fredkin circuits, respectively; however, the simulation is performed in a nondeterministic way, relying on the fact that sooner or later the simulation will choose the correct sequence of rules. Note that if the wrong rules are chosen the simulation is not aborted; the state of the system is "rolled back" so that a new nondeterministic choice can be made, hopefully the correct one. Clearly this situation could produce infinite loops; this is why one would like instead to have a *deterministic* simulation.

Here we can only give an *informal* justification for our conjecture. Notice that objects only interact indirectly, via releasing free energy units in a region or consuming them. Consider a dependency graph whose nodes are identified by object-region pairs. Two nodes are connected if the corresponding objects are present in the associated regions in some rule. A system is deterministic if no branching can be effectively used in its computations, so removing unusable rules would lead to a dependency graph where each node has out-degree at most one. Hence, any object occurring in the initial configuration of the system has some predetermined evolution path, and one of the following cases must happen:

- the path is finite, and the object evolves until there are no associated rules,

- the path leads to a cycle, and the object evolves forever (the computation yields no result),
- the evolution is “frozen” because there is not enough energy for the associated rule.

In energy-based P systems, the only way one object can influence the behavior of another object is by manipulating energy, leading to freezing or unfreezing the computational path of another object. There is no deterministic way to set an object to two different paths. If a “frozen” object receives enough energy to continue its evolution, then its computational path is the same as if it was never frozen.

So the information that can be passed from an object to another one is quite limited: giving the latter energy, as opposed to letting it freeze forever. However, every time this happens, some object must stop evolving forever. Since the initial number of objects is fixed and cannot increase, the communication complexity is bounded and this should imply non-universality.

However, even if deterministic energy-based P systems were not universal, they could nonetheless be able to simulate Fredkin gates. This should be doable if leaving some “garbage” into the system at the end of the computation is allowed. Indeed, the active objects could unfreeze the desired ones, producing the needed result. More difficult would be designing an energy-based P system that can be reused to simulate a Fredkin gate as many times as desired. We expect the reusable construction to be impossible, for the same reasons as exposed above.

Acknowledgements

The ideas exposed in this paper emerged during and after the *Tenth Brainstorming Week on Membrane Computing (BWMC 10)*, held in Seville in conjunction with the *International Conference on Developments in Membrane Computing*, from January 30th to February 3rd, 2012.

The first author gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618). The work of the other two authors was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

References

1. A. Alhazov, R. Freund, A. Leporati, M. Oswald, C. Zandron: (Tissue) P Systems with Unit Rules and Energy Assigned to Membranes. *Fundamenta Informaticae* **74**(4), 2006, 391–408.
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. R. Freund, O.H. Ibarra, Gh. Păun, H.-C. Yen: Matrix Languages, Register Machines, Vector Addition Systems. In: Proceedings of the *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–168. Available at: `http://www.gcn.us.es/3BWMC/bravolpdf/bravol155.pdf`

4. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**(1-3), 2002, 81–102.
5. P. Frisco: The Conformon-P system: A Molecular and Cell Biology-Inspired Computability Model. *Theoretical Computer Science* **312**(2-3), 2004, 295–319.
6. A. Leporati, D. Besozzi, P. Cazzaniga, D. Pescini, C. Ferretti: Computing with Energy and Chemical Reactions. *Natural Computing* **9**, 2010, 493–512.
7. A. Leporati, C. Zandron, G. Mauri: Simulating the Fredkin Gate with Energy-Based P Systems. *J Univers Comput Sci* **10**(5), 2004, 600-619.
8. A. Leporati, C. Zandron, G. Mauri: Reversible P Systems to Simulate Fredkin Circuits. *Fundamenta Informaticae* **74**, 2006, 529–548.
9. M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
10. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing* **20**(3), 2002, 295–306.
11. Gh. Păun: Computing with Membranes. *J Comput Syst Sci* **1**(61), 2000, 108-143. See also Turku Centre for Computer Science, *TUCS Report* No. **208**, 1998.
12. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
13. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
14. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
15. P Systems Webpage. `http://www.ppage.psystems.eu/`

Asynchronous and Maximally Parallel Deterministic Controlled Non-Cooperative P Systems Characterize $NFIN \cup coNFIN$

Artiom Alhazov[1,2] and Rudolf Freund[3]

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
`artiom.alhazov@unimib.it`

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

[3] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
E-mail: `rudi@emcc.at`

Summary. Membrane systems (with symbol objects) are distributed controlled multiset processing systems. Non-cooperative P systems with either promoters or inhibitors (of weight not restricted to one) are known to be computationally complete. In this paper we show that the power of the deterministic subclass of such systems is computationally complete in the sequential mode, but only subregular in the asynchronous mode and in the maximally parallel mode.

1 Introduction

The most famous membrane computing model where determinism is a criterion of universality versus decidability is the model of catalytic P systems, see [2] and [4].

It is also known that non-cooperative rewriting P systems with either promoters or inhibitors are computationally complete, [1]. Moreover, the proof satisfies some additional properties:

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

The purpose of this paper is to formally prove that computational completeness cannot be achieved by deterministic systems when working in the asynchronous or in the maximally parallel mode.

2 Definitions

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

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

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

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

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

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

Definition 1. *A* rule with context conditions $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$ *is applicable to a configuration C if r is applicable, and there exists some $j \in \{1, \cdots, m\}$ for which*

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

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

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

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

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

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

In the *sequential mode* (*sequ*), a computation step consists in the non-deterministic application of one applicable rule r, replacing its left-hand side ($lhs\,(r)$) with its right-hand side ($rhs\,(r)$). In the *maximally parallel mode* (*maxpar*), multiple applicable rules may be chosen non-deterministically to be applied in parallel to the underlying configuration to disjoint submultisets, possibly leaving some objects idle, under the condition that no further rule is applicable to them. In the *asynchronous mode* (*asyn*), any positive number of applicable rules may be chosen non-deterministically to be applied in parallel to the underlying configuration to disjoint submultisets. The computation step between two configurations C and C' is denoted by $C \Rightarrow C'$, thus yielding the binary relation $\Rightarrow: \mathbb{C}\,(O) \times \mathbb{C}\,(O)$. A computation halts when there are no rules applicable to the current configuration (*halting configuration*) in the corresponding mode.

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

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

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

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

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

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

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

Example 1. Consider an arbitrary finite set H of numbers. Choose $K = \max(H) + 1$; then we construct the following deterministic accepting P system with promoters and inhibitors:

$$\begin{aligned}
\Pi &= (O, \{a\}, s_0 f_0 \cdots f_K, R', R), \\
O &= \{a\} \cup \{s_i, f_i \mid 0 \leq i \leq K\}, \\
R' &= \{s_i \to s_{i+1} \mid 0 \leq i \leq K-1\} \cup \{f_i \to f_i \mid 0 \leq i \leq K\}, \\
R &= \{s_i \to s_{i+1}|_{a^{i+1}}, \mid 0 \leq i \leq K-1\} \\
&\cup \left\{f_i \to f_i|_{s_i, \neg a^{i+1}}, \mid 0 \leq i < K,\ i \notin H\right\} \cup \{f_K \to f_K|_{s_K}\}.
\end{aligned}$$

The system step by step, by the application of the rule $s_i \to s_{i+1}|_{a^{i+1}}$, $0 \leq i < K$, checks if (at least) $i+1$ copies of the symbol a are present. If the computation stops after i steps, i.e., if the input has consisted of exactly i copies of a, then this input is accepted if and only if $i \in H$, as exactly in this case the system does not start an infinite loop with using $f_i \to f_i|_{s_i, \neg a^{i+1}}$. If the input has contained more than $\max(H)$ copies of a, then the system arrives in the state s_K and will loop forever with $f_K \to f_K|_{s_K}$. Therefore, exactly H is accepted. To accept the complement of H instead, we simply change $i \notin H$ to $i \in H$ and as well omit the rule $f_K \to f_K|_{s_K}$. It is easy to see that for the maximally parallel mode, we can replace each rule $f_i \to f_i|_{s_i,\neg a^{i+1}}$ by the corresponding rule $f_i \to f_i|_{s_i}$; in this case, this rule may be applied with still some a being present while the system passes through the state s_i, but it will not get into an infinite loop in that case.

In sum, we have shown that

$$N_{deta}OP_1^{asyn}\left(ncoo, (pro_{1,*}, inh_{1,*})_1\right) \supseteq FIN \cup coNFIN$$

and

$$N_{deta}OP_1^{maxpar}(ncoo, pro_{1,*}) \supseteq FIN \cup coNFIN.$$

Example 2. For P systems working in the maximally parallel way we can even construct a system with inhibitors only:

$$\begin{aligned}
\Pi &= (O, \{a\}, ts_K, R)\,,\\
O &= \{a,t\} \cup \{s_i \mid 0 \le i \le K\}\,,\\
R' &= \{s_i \to ts_{i-1}, s_i \to s_i \mid 1 \le i \le K\} \cup \{t \to \lambda, s_0 \to s_0\}\,,\\
R &= \{s_i \to ts_{i-1}|_{\neg a^i} \mid 1 \le i \le K\}\\
&\cup\ \{t \to \lambda\} \cup \{s_i \to s_i|_{\neg t} \mid 0 \le i \le K,\ i \notin H\}\,.
\end{aligned}$$

This construction does not carry over to the case of the asynchronous mode, as the rule $t \to \lambda$ is applied in parallel to the rules $s_i \to ts_{i-1}|_{\neg a^i}$ until the input a^i is reached. In this case, the system canot change the state s_i anymore, and then it starts to loop if and only if $i \notin H$. To accept the complement of H instead, change $i \in H$ to $i \notin H$, i.e., in sum, we have proved that

$$N_{deta}OP_1^{maxpar}(ncoo, inh_{1,*}) \supseteq FIN \cup coNFIN.$$

As we shall show later, all the inclusions stated in Example 1 and Example 2 are equalities.

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

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

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

2.1 Register machines

In what follows we will need to simulate register machines; here we briefly recall their definition and some of their computational properties. A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, P is the set of instructions bijectively labeled by elements of B, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M can be of the following forms:

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

A register machine is *deterministic* if $l_2 = l_3$ in all its ADD instructions. A *configuration* of a register machine is described by the contents of each register and by the value of the program counter, which indicates the next instruction to be executed. Computations start by executing the first instruction of P (labeled with l_0), and terminate with reaching a $HALT$-instruction.

Register machines provide a simple universal computational model [5]. We here consider register machines used as *accepting* or as *generating* devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts having it as input. Usually, without loss of generality, we may assume that the instruction $l_h : HALT$ always appears exactly once in P, with label l_h. In the generative case, we start with empty registers and take the results of all possible halting computations.

3 Results

In this section we mainly investigate deterministic accepting P systems with context conditions and priorities on the rules (*deterministic P systems* for short) using only non-cooperative rules and working in the sequential, the asynchronous, and the maximally parallel mode.

Remark 4. We first notice that maximal parallelism in systems with non-cooperative rules means the total parallelism for all symbols to which at least one rule is applicable, and determinism guarantees that “at least one” is “exactly one” for all reachable configurations and objects. Determinism in the sequential mode requires that at most one symbol has an associated applicable rule for all reachable configurations. Surprisingly enough, in the case of the asynchronous mode we face an even worse situation than in the case of maximal parallelism – if more than one copy of a specific symbol is present in the configuration, then no rule can be applicable to such a symbol in order not to violate the condition of determinism.

We now define the *bounding* operation over multisets, with a parameter $k \in \mathbb{N}$ as follows:

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

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

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

Proof. We start by representing context conditions by predicates on boundings. Consider a rule with a simple context condition (r, p, Q), and let the current configuration be C. Then, it suffices to take $k \geq \max(|p|, \max\{|q| \mid q \in Q\})$, and let $C' = b_k(C)$. The applicability condition for (r, p, Q) may be expressed as $p \subseteq C' \wedge \left(\bigwedge_{q \in Q} q \not\subseteq C'\right)$. Indeed, $x \subseteq C \longleftrightarrow x \subseteq C'$ for every multiset x with $|x| \leq k$, because for every $a \in O$, $|x|_a \leq |C|_a \longleftrightarrow |x|_a \leq \min(|C|_a, k)$ holds if $|x|_a \leq k$. Finally, we notice that context conditions which are not simple can be represented by a disjunction of the corresponding predicates.

Conversely, we show that any predicate $E \subseteq \{0, \cdots, k\}^O$ for the bounding mapping b_k for rule r can be represented by some context conditions. For each multiset $c \in E$, we construct a simple context condition to the effect of "contains c, but, for each a contained in c for less than k times, not more than $|c|_a$ symbols a":

$$\left\{\left(r, c, \left\{a^{|c|_a+1} \mid |c|_a < k\right\}\right) \mid c \in E\right\}.$$

Joining multiple simple context conditions over the same rule into one rule with context conditions concludes the proof. □

The following theorem is valid even when the rules are not restricted to non-cooperative ones, and when determinism is not required, in either derivation mode (also see [3]).

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

Proof. A rule is prohibited from being applicable due to a priority relation if and only if at least one of the rules with higher priority might be applied. Let r be a rule of a P system $(O, \Sigma, w, R', R, >)$, and let $r_1 > r, \cdots, r_n > r$. Hence, the rule r is not blocked by the rules $r_1, \cdots, r_n$ if and only if the left-hand sides of the rules $r_1, \cdots, r_n$, $lhs(r_1), \cdots, lhs(r_n)$ are not present in the current configuration or the context conditions given in these rules are not fulfilled. According to Lemma 1, these context conditions can be formulated as predicates on the bounding b_k where k is the maximum of weights of all left-hand sides, promoters, and inhibitors in the rules with higher priority $r_1, \cdots, r_n$. Together with the context conditions from r itself, we finally get context conditions for a new rule r' simulating r, but also incorporating the conditions of the priority relation. Performing this transformation for all rules r concludes the proof. □

Remark 5. From [3] we already know that in the case of rules without context conditions, the context conditions in the new rules are only sets of atomic inhibitors, which also follows from the construction given above. A careful investigation of the construction given in the proof of Theorem 1 reveals the fact that the maximal weights for the promoters and inhibitors to be used in the new system are bounded by the number k in the bounding b_k.

3.1 Sequential Systems

Although throughout the rest of the paper we are not dealing with sequential systems anymore, the proof of the following theorem gives us some intuition why, for deterministic non-cooperative systems, there are severe differences between the sequential mode and the asynchronous or the maximally parallel mode.

Theorem 2. $N_{deta}OP_1^{sequ}\left(ncoo, pro_{1,1}, inh_{1,1}\right) = NRE$.

Proof. Consider an arbitrary deterministic register machine $M = (m, B, l_0, l_h, P)$. We simulate M by a deterministic P system $\Pi = \left(O, \{a_1\}, l_0, R\right)$ where

$$\begin{aligned} O &= \{a_j \mid 1 \leq j \leq m\} \cup \{l, l_1, l_2 \mid l \in B\}, \\ R &= \{l \to a_j l' \mid (l : ADD(j), l') \in P\} \\ &\cup \{l \to l_1|_{a_j}, a_j \to a_j'|_{l_1, \neg a_j'}, l_1 \to l_2|_{a_j'}, a_j' \to \lambda|_{l_2}, l_1 \to l'|_{\neg a_j'}, \\ &\quad l \to l''|_{\neg a_j} \mid (l : SUB(j), l', l'') \in P\}. \end{aligned}$$

We claim that Π is deterministic and non-cooperative, and it accepts the same set as M. □

As can be seen in the construction of the deterministic P system in the proof above, the rule $a_j \to a_j'|_{l_1, \neg a_j'}$ used in the sequential mode can be applied exactly once, priming exactly one symbol a_j to be deleted afterwards. Intuitively, in the asynchronous or the maximally parallel mode, it is impossible to choose only one symbol out of an unbounded number of copies to be deleted. The bounding operation defined above will allow us to put this intuition into a formal proof.

3.2 Asynchronous and Maximally Parallel Systems

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

Lemma 2. *Assume $u \Rightarrow x$ and $v \Rightarrow y$. Then $b_k(u) = b_k(v)$ implies $b_k(x) = b_k(y)$.*

Proof. Equality $b_k(u) = b_k(v)$ means that for every symbol $a \in O$, if $|u|_a \neq |v_a|$ then $|u|_a \geq k$ and $|v|_a \geq k$, and we have a few cases to be considered. If no rule is applicable to a, then the inequality of symbols a will be indistinguishable after bounding also in the next step (both with at least k copies of a). Otherwise, exactly one rule r is applicable to a (by determinism, and bounding does not affect applicability), then the difference of the multiplicities of the symbol a may only lead to differences of the multiplicities of symbols b for all $b \in rhs(r)$. However, either all copies of a are erased by the rule $a \to \lambda$ or else at least one copy of a symbol b will be generated from each copy of a by this rule alone, so $|x|_b \geq |u|_a \geq k$ and $|y|_b \geq |v|_a \geq k$, so all differences of multiplicities of an object b in u and v will be indistinguishable after bounding in this case, too. □

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

Proof. Let w be the fixed part of the initial configuration. Then we consider computations from uw and from vw. Clearly, $b_k(uw) = b_k(vw)$. Equality of boundings is preserved by one computation step, and hence, by any number of computation steps.

Assume the contrary of the claim: one of the computations halts after s steps, while the other one does not, i.e., let $uw \Rightarrow^s u'$ and $vw \Rightarrow^s v'$. By the previous paragraph, $b_k(u') = b_k(v')$. Since bounding does not affect applicability of rules, either both u' and v' are halting, or none of them. The contradiction proves the claim. □

We should like to notice that the arguments in the proofs of Lemma 2 and Corollary 1 are given for the maximal parallel mode; following the observation stated at the end of Remark 4, these two results can also be argued for the asynchronous mode.

Theorem 3. *For deterministic P systems working in the asynchronous or in the maximally parallel mode, we have the following characterization:*

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

Proof. Each equivalence class induced by bounding is completely accepted or completely rejected. If no infinite equivalence class is accepted, then the accepted set is finite (containing numbers not exceeding $(k-1) \cdot |O|$). If at least one infinite equivalence class is accepted, then the rejected set is finite (containing numbers not exceeding $(k-1) \cdot |O|$). This proves the "at most $NFIN \cup coNFIN$" part.

In Examples 1 and 2 we have already shown that

$$N_{deta}OP_1^{\alpha}(ncoo, pro_{1,*}, inh_{1,*}) \supseteq FIN \cup coNFIN$$

for $\alpha \in \{asyn, maxpar\}$ as well as

$$N_{deta}OP_1^{maxpar}\left(ncoo, \gamma_{1,*}\right) \supseteq FIN \cup coNFIN$$

for $\gamma \in \{pro, inh\}$. This observation concludes the proof. □

There are several questions remaining open: First of all, we do not know whether inhibitors in the rules are sufficient to yield $FIN \cup coNFIN$ with the asynchronous mode, too. Moreover, it would be interesting to see if the parameter K used in the proof of the preceding theorem induces an infinite hierarchy on the families $N_{deta}OP_1^{\alpha}\left(ncoo, \gamma_{1,K}\right)$, $\alpha \in \{asyn, maxpar\}$, $\gamma \in \{pro, inh\}$.

4 Conclusion

We have shown that, like in case of catalytic P systems, for non-cooperative P systems with promoters and/or inhibitors (with or without priorities), determinism is a criterion drawing a borderline between universality and decidability. In fact, for non-cooperative P systems working in the maximally parallel or the asynchronous mode, we have computational completeness in the unrestricted case, and only all finite number sets and their complements in the deterministic case.

Acknowledgements. The first author gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618).

References

1. A. Alhazov, D. Sburlan: Ultimately Confluent Rewriting Systems. Parallel Multiset-Rewriting with Permitting or Forbidding Contexts. *Membrane Computing, 5th International Workshop* WMC 2004, Milano, Revised Selected and Invited Papers (G. Mauri et al., eds.), LNCS 3365, Springer, 2005, 178–189.
2. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient, *Theoretical Computer Science* **330**, 2, 2005, 251–266.
3. R. Freund, M. Kogler, M. Oswald, A General Framework for Regulated Rewriting Based on the Applicability of Rules. In: J. Kelemen, A. Kelemenová, *Computation, Cooperation, and Life*, Springer, LNCS 6610, 2011, 35–53.
4. O.H. Ibarra, H.-C. Yen: Deterministic Catalytic Systems are Not Universal, *Theoretical Computer Science* **363**, 2006, 149–161.
5. M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
6. Gh. Păun: *Membrane Computing. An Introduction*, Springer, 2002.
7. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
8. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
9. P systems webpage. `http://ppage.psystems.eu`

The Computational Power of Exponential-Space P Systems with Active Membranes

Artiom Alhazov[1,2], Alberto Leporati[1], Giancarlo Mauri[1], Antonio E. Porreca[1], and Claudio Zandron[1]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Universit degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
artiom.alhazov@unimib.it
{leporati,mauri,porreca,zandron}@disco.unimib.it

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

Summary. We show that exponential-space P systems with active membranes characterize the complexity class **EXPSPACE**. This result is proved by simulating Turing machines working in exponential space via uniform families of P systems with restricted elementary active membranes; the simulation is efficient, in the sense that the time and space required are at most polynomial with respect to the resources employed by the simulated Turing machine.

1 Introduction

P systems with active membranes have been introduced in [9] as a variant of P systems where the membranes have an active role during computations: they have an electrical charge that can inhibit or activate the rules that govern the evolution of the system, and they can grow in number by using division rules.

In several papers these systems were used to attack computationally hard problems, by exploiting the possibility to create, in polynomial time, an exponential number of membranes that evolve in parallel. Hence, for instance, it has been proved that P systems with active membranes can solve **PSPACE**-complete problems [11, 2] in polynomial time. When division rules operate only on *elementary* membranes (i.e. membranes not containing other membranes), such systems are still able to efficiently solve **NP**-complete problems [12, 5]. More recently, in [7] it was proved that all problems in $\mathbf{P}^{\mathbf{PP}}$ (a possibly larger class including the polynomial hierarchy) can also be solved in polynomial time using P systems with elementary membrane division. On the other hand, if division of membranes is not allowed then the efficiency apparently decreases [12]: no **NP**-complete problem can be solved in polynomial time without using division rules unless $\mathbf{P} = \mathbf{NP}$ holds.

A measure of space complexity for P systems has been introduced [6] in order to analyze the time-space trade-off exploited when P systems are used to efficiently solve computationally hard problems. The space required by a P system is the maximal size it can reach during any computation, defined as the sum of the number of membranes and the number of objects. A uniform family $\boldsymbol{\Pi}$ of recognizer P systems is said to solve a problem in space $f\colon \mathbb{N} \to \mathbb{N}$ if no P system in $\boldsymbol{\Pi}$ associated to an input string of length n requires more than $f(n)$ space. Under this notion of space complexity, in [8] it has been proved that the class of problems solvable in polynomial space by P systems with active membranes, denoted by $\mathbf{PMCSPACE}_{\mathcal{AM}}$, coincides with **PSPACE**. This result is proved by mutual simulation of P systems and Turing machines.

The techniques used up to now to simulate a polynomial-space Turing machine via a polynomial-space family of P systems [7] do not seem to apply when the space bound is less strict, i.e., exponential or even super-exponential. Indeed, we would need P systems with an exponential number of membranes with distinct labels, and such systems cannot be built in a polynomial number of steps by a deterministic Turing machine (as required by the notion of polynomial-time uniformity usually employed in the literature [5]).

Here we show that, by using different techniques, exponential-space Turing machines can be simulated by exponential-space P systems; hence, the classes of problems solvable by P systems with active membranes and by Turing machines in exponential space coincide; in symbols, $\mathbf{EXPMCSPACE}_{\mathcal{AM}} = \mathbf{EXPSPACE}$.

The rest of the paper is organized as follows. In section 2 we recall some definitions concerning P systems with active membranes and their space complexity. In section 3 we describe how P systems with restricted elementary membranes can be used to simulate Turing machines; an analysis on the resources (time and space) needed to perform this simulation is also given. Section 4 contains the statement of our characterization of **EXPSPACE**, while section 5 provides the conclusions as well as some directions for further research.

2 Definitions

We assume the reader to be familiar with the basic terminology and results concerning P systems with active membranes (see [10], chapters 11–12 for a survey). Here we just recall some definitions that are relevant for the results presented in this paper.

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

- *Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- *Λ is a finite set of labels for the membranes;*

- *μ is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of d membranes enumerated by $1, \dots, d$; furthermore, each membrane is labeled by an element of Λ, not necessarily in a one-to-one way;*
- *$w_1, \dots, w_d$ are strings over Γ, describing the initial multisets of objects placed in the d regions of μ;*
- *R is a finite set of rules.*

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge* (or polarization), which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

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 charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by 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 charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β.
- *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 charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labeled by h, having charge α and containing an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b.
- *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 charge α, containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and charge 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

$$[[\;]_{h_1}^+ \cdots [\;]_{h_k}^+ [\;]_{h_{k+1}}^- \cdots [\;]_{h_n}^-]_h^\alpha \to [[\;]_{h_1}^\delta \cdots [\;]_{h_k}^\delta]_h^\beta [[\;]_{h_{k+1}}^\epsilon \cdots [\;]_{h_n}^\epsilon]_h^\gamma$$

  They can be applied to a membrane labeled by h, having charge α, containing the positively charged membranes $h_1, \dots, h_k$, the negatively charged membranes $h_{k+1}, \dots, h_n$, and possibly some neutral membranes. The membrane h is divided into two copies having charge β and γ, respectively; the positive children are placed inside the former membrane, their charge changed to δ, while the negative ones are placed inside the latter membrane, their charges changed to ϵ. Any neutral membrane inside h is duplicated and placed inside both copies.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane any number of evolution rules can be applied simultaneously).
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same reasoning applies to each membrane that can be involved to communication, dissolution, elementary or non-elementary division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- While all the chosen rules are considered to be applied simultaneously during each computation step, they are logically applied in a bottom-up fashion: first, all evolution rules are applied to the elementary membranes, then all communication, dissolution and division rules; then the application proceeds towards the root of the membrane structure. In other words, each membrane evolves only after its internal configuration has been updated.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

The precise variant of P systems we use in this paper does not use dissolution or non-elementary division rules.

Definition 2. *A* P system with restricted elementary active membranes *is a P system with active membranes where only object evolution, send-in, send-out, and elementary division rules are used. This kind of P systems is denoted by* $\mathcal{AM}(-\mathrm{d}, -\mathrm{n})$.

A *halting computation* of the P system $\mathit{\Pi}$ is a finite sequence of configurations $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable by $\mathcal{C}_i$ via a single computation step, and no rules can be applied anymore in $\mathcal{C}_k$. A *non-halting* computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as *recognizers* by employing two distinguished objects YES and NO; exactly one of these must be sent out from the outermost membrane during each computation, in order to signal acceptance or rejection respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists. All P systems in this paper are confluent.

In order to solve decision problems (i.e., decide languages), we use *families* of recognizer P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [3].

Definition 3. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* (polynomial-time) uniform *if the mapping* $x \mapsto \Pi_x$ *can be computed by two deterministic polynomial-time Turing machines* F *(for "family") and* E *(for "encoding") as follows:*

- *The machine* F*, taking as input* the length n of x *in unary notation, constructs a P system* Π_n*, which is common for all inputs of length* n*, with a distinguished input membrane.*
- *The machine* E*, on input* x*, outputs a multiset* w_x *(an encoding of the specific input* x*).*
- *Finally,* Π_x *is simply* Π_n *with* w_x *added to the multiset placed inside its input membrane.*[3]

Definition 4. *If the mapping* $x \mapsto \Pi_x$ *is computed by a* single *polynomial-time Turing machine, the family* $\boldsymbol{\Pi}$ *is said to be* semi-uniform*. In this case, inputs of the same size may be associated with P systems having possibly different membrane structures and rules.*

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [3] for further details on the encoding of P systems.

[3] Notice that this definition of uniformity is (possibly) weaker than the other one commonly used in membrane computing [5], where the Turing machine F maps each input x to a P system $\Pi_{s(x)}$, where $s\colon \Sigma^\star \to \mathbb{N}$ is a measure of the size of the input; in our case, $s(x)$ is always $|x|$.

Finally, we describe how space complexity for families of recognizer P systems is measured, and the related complexity classes [6].

Definition 5. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and the total number of objects they contain. If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a halting computation of Π, then the* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

$$|\boldsymbol{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$$

or, in the case of a non-halting computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$,

$$|\boldsymbol{\mathcal{C}}| = \sup\{|\mathcal{C}_i| : i \in \mathbb{N}\}.$$

Non-halting computations might require an infinite amount of space (in symbols $|\boldsymbol{\mathcal{C}}| = \infty$): for example, if the number of objects strictly increases at each computation step.

The space required by Π *itself is then*

$$|\Pi| = \sup\{|\boldsymbol{\mathcal{C}}| : \boldsymbol{\mathcal{C}} \textit{ is a computation of } \Pi\}.$$

Notice that $|\Pi| = \infty$ might occur if either Π has a non-halting computation requiring infinite space (as described above), or Π has an infinite set of halting computations, such that for each bound $b \in \mathbb{N}$ there exists a computation requiring space larger than b.

Finally, let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems, and let $f \colon \mathbb{N} \to \mathbb{N}$. We say that $\boldsymbol{\Pi}$ operates within space bound f *iff $|\Pi_x| \leq f(|x|)$ for each $x \in \Sigma^\star$.*

By $\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$ we denote the class of languages which can be decided by uniform families of confluent P systems of type $\mathcal{D}$ where each $\Pi_x \in \boldsymbol{\Pi}$ operates within space bound $f(|x|)$. The class of languages decidable in exponential space by uniform families of P systems of type $\mathcal{D}$ is denoted by $\mathbf{EXPMCSPACE}_{\mathcal{D}}$, while the corresponding class for semi-uniform families is $\mathbf{EXPMCSPACE}^\star_{\mathcal{D}}$. The classes defined in terms of non-confluent P systems are denoted by $\mathbf{NEXPMCSPACE}_{\mathcal{D}}$ and $\mathbf{NEXPMCSPACE}^\star_{\mathcal{D}}$, respectively.

For the precise definitions and properties of Turing machines and, in particular, the space complexity classes **PSPACE** and **EXPSPACE**, we refer the reader to [4].

3 Simulating a Turing machine

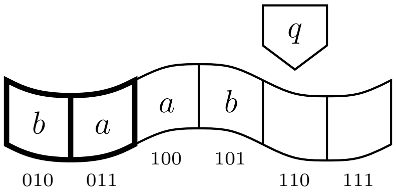
In this section we show that exponential-space deterministic Turing machines can be simulated by P systems with restricted elementary active membranes with a polynomial slowdown and a polynomial growth in space.

Theorem 1. *Let M be a single-tape deterministic Turing machine working in time $t(n)$ and space $s(n)$, where $s(n) \leq n + 2^{p(n)}$ for some polynomial p. Then there exists a uniform family of confluent P systems with restricted elementary active membranes $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ operating in time $O\big(t(n)s(n)\log s(n)\big)$ and space $O\big(s(n)\log s(n)\big)$ such that $L(\boldsymbol{\Pi}) = L(M)$.*

We describe how the simulation is carried out by examining a specific example, and generalizing from there. Let M be a Turing machine having tape alphabet $\Gamma = \{a, b, \sqcup\}$, where $\sqcup$ denotes a blank tape cell, and using space $n + 2^n$ (i.e., we choose $p(n) = n$). Also let Q be the set of non-final states of M, and

$$\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright\}$$

its transition function. Assume that M processes the input $x = ba$ of length 2: then M uses a total of $2 + 2^2 = 6$ tape cells. Suppose that, after a few computation steps, M reaches the following configuration:

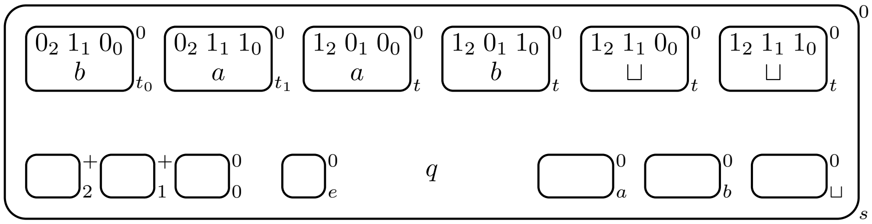


that is, the state of M is q, the tape contains the string $baab$ followed by two blank cells, and the tape head is located on the fifth cell. The picture also shows (in binary) the non-standard numbering scheme for tape cells that we employ:

- The first n cells, that initially contain the input (highlighted by a thick border), are denoted by $2^{p(n)} - n, \ldots, 2^{p(n)} - 1$ (e.g., 010 and 011 in our example). These numbers, when written in binary over $p(n) + 1$ bits, all have 0 as their most significant bit.
- The remaining $2^{p(n)}$ cells are denoted by $2^{p(n)}, \ldots, 2 \times 2^{p(n)} - 1$ (e.g., 100 to 111 in our example). These numbers, when written in binary over $p(n) + 1$ bits, all have 1 as their most significant bit.

3.1 Representing the configuration of the Turing machine

The configuration of M described above is encoded in the following configuration $\mathcal{C}_1$ of the P system Π_x simulating it (how this configuration of Π_x is reached from its initial configuration will be described later):



Inside the outermost membrane, labeled by s, we have $n + 2^{p(n)}$ membranes (6 in our example) representing the tape cells of M; n of them are labeled $t_0, \dots, t_{n-1}$, and the remaining ones (which are generated by membrane division, as described below) by t. We refer to these membranes as *tape-membranes*. Each tape-membrane contains two pieces of information: a set of $p(n)+1$ (3 in our example) subscripted bits, the *bit-objects*, encoding the number of the tape cell of M it represents (the subscript are used to preserve the order of the bits), and an object taken from the alphabet of M, denoting the symbol written in that tape cell (the *symbol-object*). For instance, the membrane $[1_2 0_1 1_0 b]_t^0$ corresponds to tape cell 101, which contains the symbol b.

The state of M is represented by a *state-object* (q in the example), which will regulate the simulation of each computation step of M. At the beginning of the simulation of each computation step of M, the state-object resides in membrane s.

On the lower-left side of the picture we have $p(n) + 1$ membranes, called *position-membranes* and labeled by $p(n), \dots, 0$, whose electrical charge encodes in binary the current position of the tape head of M; here a positive charge represents a 1 bit, while a neutral charge denotes 0. For instance, in the picture we have $[\,]_2^+ [\,]_1^+ [\,]_0^0$ representing position 110.

The auxiliary membrane labeled by e, the *error-membrane*, will have its charge set to positive whenever Π_x nondeterministically chooses a “wrong” computation path while simulating a computation step of M (see below).

Finally, on the lower-right side of the picture, we have membranes labeled by symbols from the alphabet of M (the *symbol-membranes*). These will be used, once again by setting their charge, to read the symbol currently under the tape head of M.

We shall now describe how to simulate a computation step of M starting from its current configuration, as encoded by Π_x. Later on we will describe how the configuration of Π_x representing the initial configuration of M can be obtained.

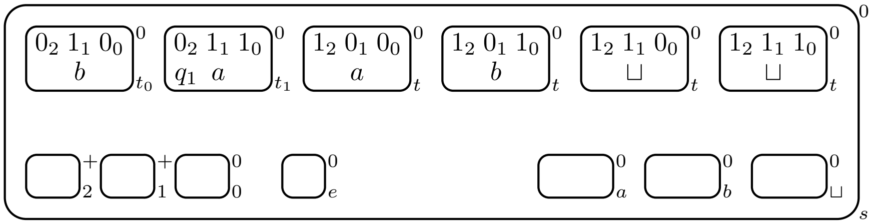
3.2 Simulating a computation step of *M*

In order to simulate a computation step of M, we need to identify which symbol is located under its tape head; note that the state q is already stored in the state-object. Since most of the tape-membranes of Π_x have the same label t (and those labeled by $t_0, \dots, t_{n-1}$ behave the same way in this phase, i.e., have the same associated set of rules) there is no way to identify the correct tape-membrane from the outside. Hence, we shall *guess* the tape-membrane corresponding to the cell under the head, then check if selected the right one.

This "guessing" is performed by the state-object, which nondeterministically enters one of the tape membranes using one of the following rules:

$$q\,[\;]_h^0 \rightarrow [q_1]_h^0 \qquad \text{for } q \in Q \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}.$$

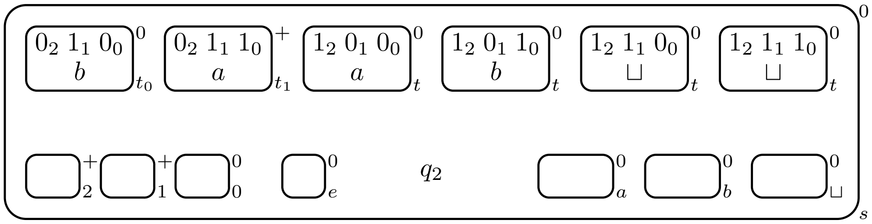
First, suppose q enters the wrong membrane, e.g., 011 instead of 110, producing the following configuration:



The state-object q_1 is immediately sent back out, changing the charge of the membrane to positive using one of the rules

$$[q_1]_h^0 \rightarrow [\;]_h^+\, q_2 \qquad \text{for } q \in Q \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}.$$

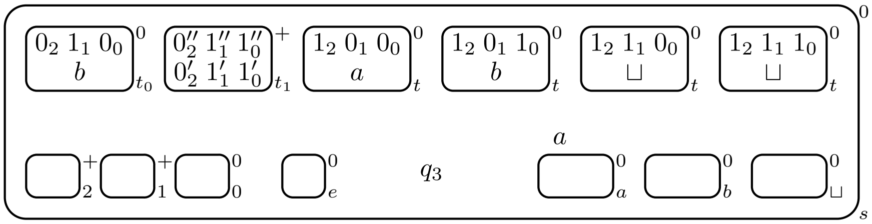
Note that there will always be at most one positive tape-membrane, i.e., the membrane being checked at the current time.



When a tape-membrane is positive, the symbol-object it contains (a in the example) is sent out, while the bit-objects are replicated in a primed and a double-primed versions. At the same time, the state-object waits by increasing its subscript (such waiting steps will be implicit from now on). The corresponding rules are

$$\begin{array}{ll}
[\gamma]_h^+ \rightarrow [\;]_h^+\, \gamma & \text{for } \gamma \in \Gamma \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} \\
[0_i \rightarrow 0'_i 0''_i]_h^+ & \text{for } 0 \leq i \leq p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} \\
[1_i \rightarrow 1'_i 1''_i]_h^+ & \text{for } 0 \leq i \leq p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} \\
[q_2 \rightarrow q_3]_s^0 & \text{for } q \in Q.
\end{array}$$

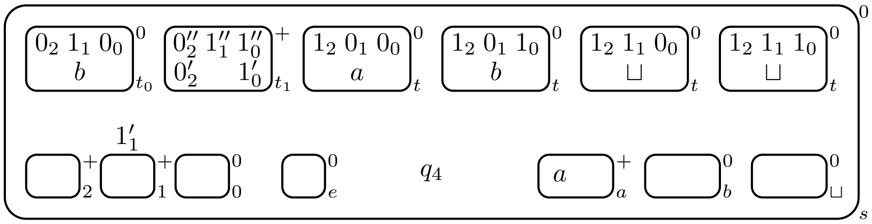
In our example, we obtain the following configuration:

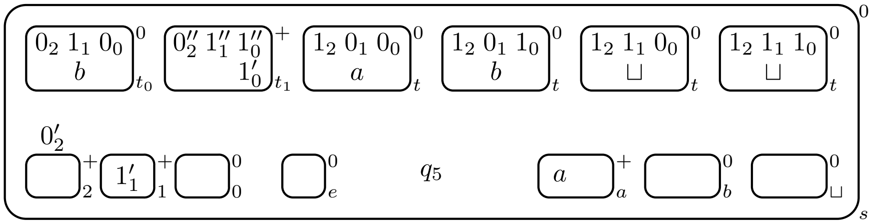

$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ $0'_2\ 1'_1\ 1'_0$ t_1 $+$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ $\sqcup$ t 0
$1_2\ 1_1\ 1_0$ $\sqcup$ t 0
2 $+$
1 $+$
0 0
e 0
q_3
a
a 0
b 0
$\sqcup$ 0
s 0

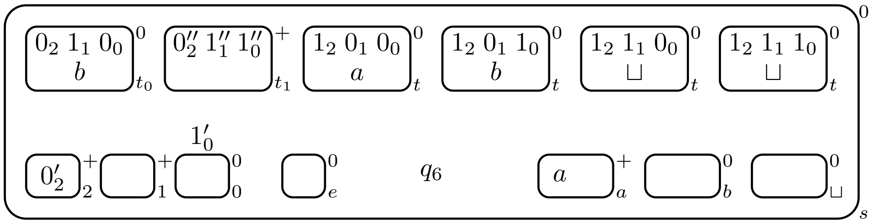
Now, the symbol-object is sent to the corresponding symbol-membrane, setting its charge to positive (thus allowing the state-object to identify the symbol under the tape head). At the same time, the primed bit-objects inside the positive tape-membrane will be sent (in nondeterministic order) to the corresponding position-membranes and compared with their charge. In our example we have $0'_2 1'_1 1'_0$ and $[\ ]_2^+ [\ ]_1^+ [\ ]_0^0$ (where the most significant and the least significant bits differ). If there is a mismatch on a certain bit, the corresponding bit-object will produce an error-object e, otherwise it will be deleted. The error-objects will set the charge of the error-membrane to positive, so that the state-object may identify the error when all comparisons have been made, and will be in turn deleted. This phase, whose duration is $p(n) + 4$ steps, involves the following rules:

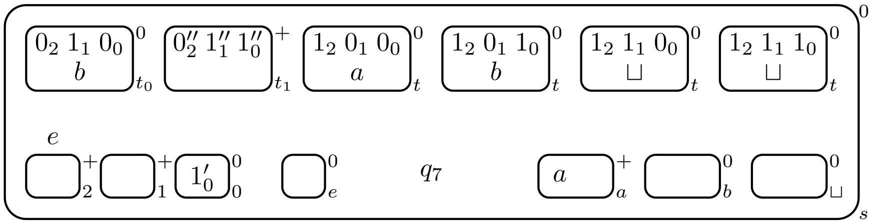
$$\begin{array}{ll}
\gamma\ [\ ]_\gamma^0 \to [\gamma]_\gamma^+ & \text{for } \gamma \in \varGamma \\
[0'_i]_h^+ \to [\ ]_h^+\ 0'_i & \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} \\
[1'_i]_h^+ \to [\ ]_h^+\ 1'_i & \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} \\
0'_i\ [\ ]_i^\alpha \to [0'_i]_i^\alpha & \text{for } 0 \le i \le p(n) \text{ and } \alpha \in \{0, +\} \\
1'_i\ [\ ]_i^\alpha \to [1'_i]_i^\alpha & \text{for } 0 \le i \le p(n) \text{ and } \alpha \in \{0, +\} \\
[0'_i \to \lambda]_i^0 & \text{for } 0 \le i \le p(n) \\
[1'_i \to \lambda]_i^+ & \text{for } 0 \le i \le p(n) \\
[0'_i]_i^+ \to [\ ]_i^+\ e & \text{for } 0 \le i \le p(n) \\
[1'_i]_i^0 \to [\ ]_i^0\ e & \text{for } 0 \le i \le p(n) \\
e\ [\ ]_e^\alpha \to [e]_e^+ & \text{for } \alpha \in \{0, +\} \\
[e \to \lambda]_e^+ & \\
[q_j \to q_{j+1}]_s^0 & \text{for } 3 \le j \le p(n) + 6 \text{ and } q \in Q.
\end{array}$$

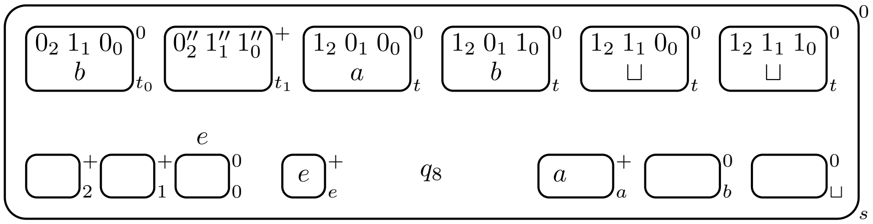
In our example, the computation may proceed as follows (for some concrete nondeterministic choices in the order the bit-objects are sent out).

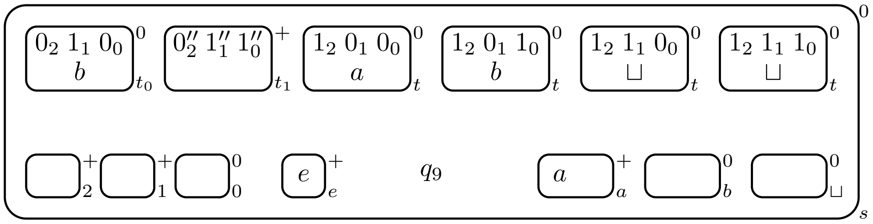

$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ $0'_2\ 1'_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
$1'_1$
2 +
1 +
0 0
e 0
q_4
a a +
b 0
⊔ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ $1'_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
$0'_2$
2 +
$1'_1$ 1 +
0 0
e 0
q_5
a a +
b 0
⊔ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
$0'_2$ 2 +
1 +
$1'_0$
0 0
e 0
q_6
a a +
b 0
⊔ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
e
2 +
1 +
$1'_0$ 0 0
e 0
q_7
a a +
b 0
⊔ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
2 +
1 +
e
0 0
e e +
q_8
a a +
b 0
⊔ 0
s 0

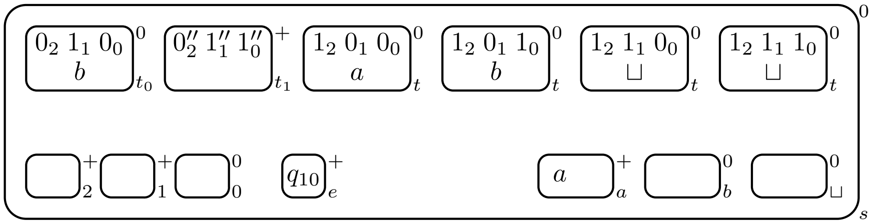

$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
2 +
1 +
0 0
e e +
q_9
a a +
b 0
⊔ 0
s 0

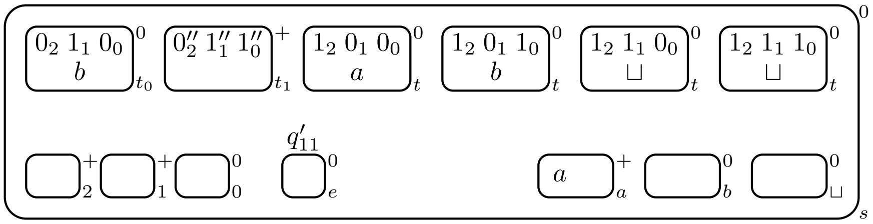
While any remaining error-object is deleted, the state-object may now enter the error-membrane in order to check if a bit mismatch has been found (thus, if the system chose the wrong tape-membrane). It is sent out in a primed version if this is the case, while simultaneously resetting the charge of e to neutral. We use the following rules:

$$q_{p(n)+7}\ [\ ]_e^{\alpha} \rightarrow [q_{p(n)+8}]_e^{\alpha} \qquad \text{for } q \in Q \text{ and } \alpha \in \{0,+\}$$

$$[q_{p(n)+8}]_e^{+} \rightarrow [\ ]_e^{0}\ q'_{p(n)+9} \qquad \text{for } q \in Q.$$

In our example, these two steps produce the following configurations:

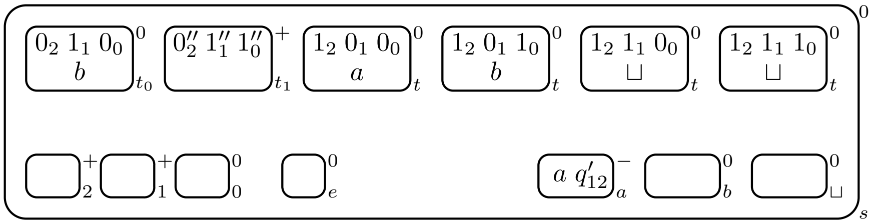

$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
2 +
1 +
0 0
q_{10} e +
a a +
b 0
⊔ 0
s 0

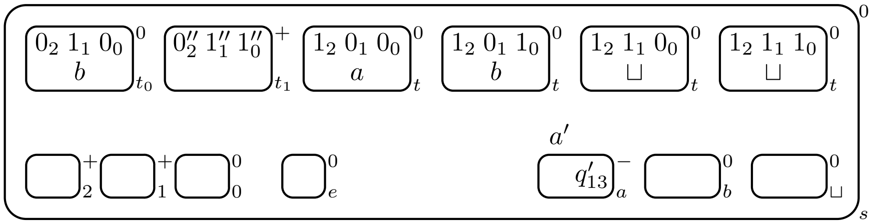

$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
2 +
1 +
0 0
q'_{11}
e 0
a a +
b 0
⊔ 0
s 0

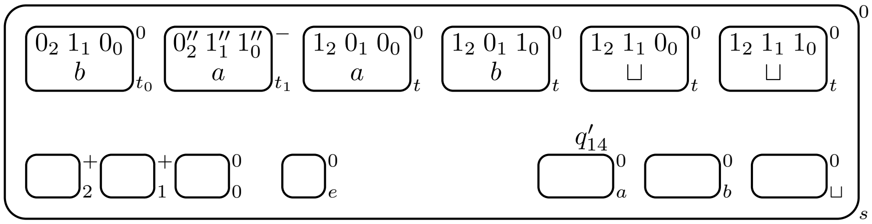
Having guessed the wrong tape-membrane, the system must now send the symbol-object back to its original tape-membrane (the only positively charged one); it does so by setting the charge of the symbol-membrane to negative. In the subsequent three steps, the configuration of Π_x is reset to $\mathcal{C}_1$, *except that the tape-membrane we chose is set to negative*. This requires the following rules:

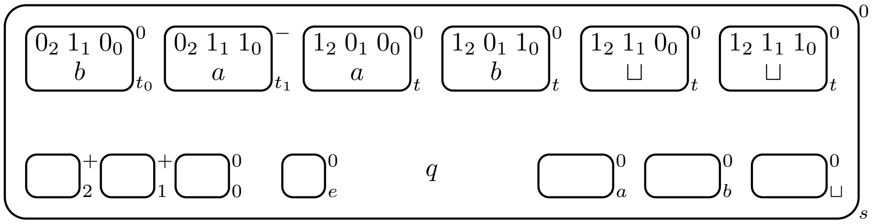
$$\begin{array}{lll}
q'_{p(n)+9}\ [\ ]_\gamma^+ \to [q'_{p(n)+10}]_\gamma^- & \text{for } q \in Q \text{ and } \gamma \in \Gamma & \\
[\gamma]_\gamma^- \to [\ ]_\gamma^-\ \gamma' & \text{for } \gamma \in \Gamma & \\
[q'_{p(n)+10} \to q'_{p(n)+11}]_\gamma^- & \text{for } q \in Q \text{ and } \gamma \in \Gamma & \\
\gamma'\ [\ ]_h^+ \to [\gamma]_h^- & \text{for } \gamma \in \Gamma \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} & \\
[q'_{p(n)+11}]_\gamma^- \to [\ ]_\gamma^0\ q'_{p(n)+12} & \text{for } q \in Q \text{ and } \gamma \in \Gamma & \\
[0''_i \to 0_i]_h^- & \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} & (1) \\
[1''_i \to 1_i]_h^- & \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} & (2) \\
[q'_{p(n)+12} \to q]_s^0 & \text{for } q \in Q. &
\end{array}$$

In the example, we obtain the following sequence of configurations:


$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
2 + 1 + 0 0 e 0
$a\ q'_{12}$ a −
b 0
⊔ 0
s 0

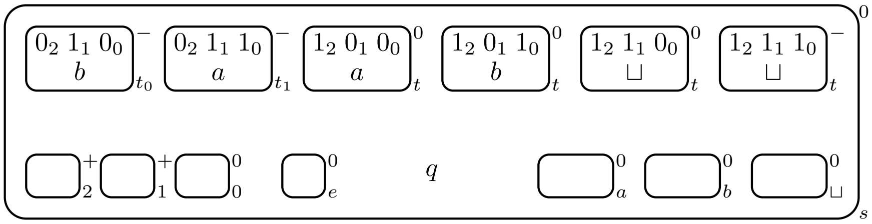

$0_2\ 1_1\ 0_0$ b t_0 0
$0''_2\ 1''_1\ 1''_0$ t_1 +
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
2 + 1 + 0 0 e 0
a'
q'_{13} a −
b 0
⊔ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0_2''\ 1_1''\ 1_0''$ a t_1 −
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
q_{14}'
2 + 1 + 0 0 e 0 a 0 b 0 ⊔ 0
s 0

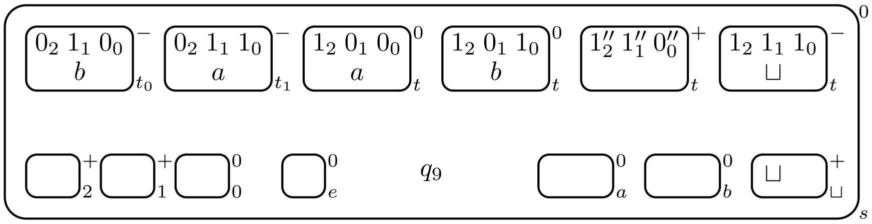

$0_2\ 1_1\ 0_0$ b t_0 0
$0_2\ 1_1\ 1_0$ a t_1 −
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t 0
q
2 + 1 + 0 0 e 0 a 0 b 0 ⊔ 0
s 0

The system can now guess another tape-membrane, repeating the previous steps while making wrong guesses and thus increasing the number of negatively charged tape-membranes (which are ignored during the guessing step). After at most $n + 2^{p(n)} - 1$ wrong guesses (e.g., 5 guesses in our example), the system finally chooses the correct tape-membrane.

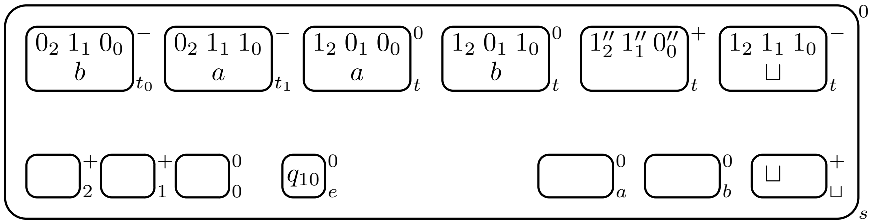
For instance, suppose Π_x made the three consecutive wrong guesses 011, 111 and 010, thus reaching the following configuration:


$0_2\ 1_1\ 0_0$ b t_0 −
$0_2\ 1_1\ 1_0$ a t_1 −
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ ⊔ t 0
$1_2\ 1_1\ 1_0$ ⊔ t −
q
2 + 1 + 0 0 e 0 a 0 b 0 ⊔ 0
s 0

Now assume that the state-object finally enters the correct membrane 110. The bit-checking phase proceeds as described above for $p(n) + 7$ steps (i.e., 9 steps in our case), making Π_x reach the following configuration:


$0_2\ 1_1\ 0_0$ b t_0 −
$0_2\ 1_1\ 1_0$ a t_1 −
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2''\ 1_1''\ 0_0''$ t +
$1_2\ 1_1\ 1_0$ ⊔ t −
q_9
2 + 1 + 0 0 e 0 a 0 b 0 ⊔ ⊔ +
s 0

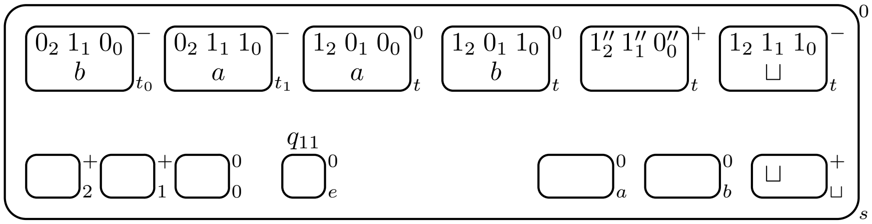
In this configuration the error-membrane is neutral, as all bit-objects match the corresponding position-membranes. The state-object enters membrane e



but this time, since the error-membrane is neutral, it is sent out in a non-primed version, using the rule

$$[q_{p(n)+8}]_e^+ \to [\ ]_e^0\ q_{p(n)+9} \qquad\qquad \text{for } q \in Q$$

thus producing the configuration



As before, the state-object is sent to the only positively charged symbol-membrane, but this time it sets it to neutral:

$$q_{p(n)+9}\ [\ ]_\gamma^+ \to [q_{p(n)+10}]_\gamma^0 \qquad\qquad \text{for } q \in Q \text{ and } \gamma \in \Gamma.$$

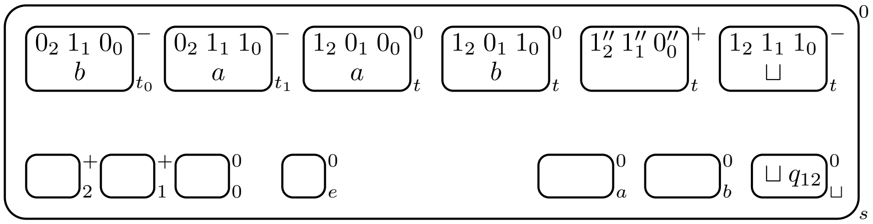
The symbol-object inside responds to this change of charge by deleting itself,

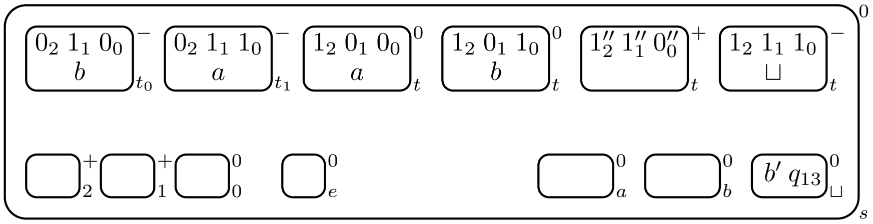
$$[\gamma \to \lambda]_\gamma^0 \qquad\qquad \text{for } \gamma \in \Gamma$$

while at the same time the state-object produces the primed version of the new symbol-object corresponding to the symbol written by the Turing machine. Assume that the transition function of M specifies that $\delta(q, \sqcup) = (r, b, \triangleleft)$; the corresponding rules are

$$[q_{p(n)+10} \to q_{p(n)+11}\ \sigma']_\gamma^0 \quad \text{if } \delta(q,\gamma) = (r,\sigma,d) \text{ for some } r \in Q,\ \sigma \in \Gamma,\ d \in \{\triangleleft, \triangleright\}.$$

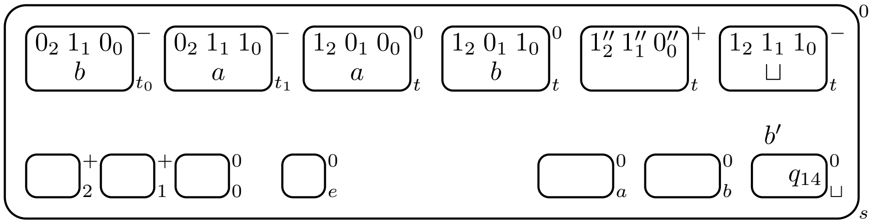
Hence, our example configuration evolves as follows:

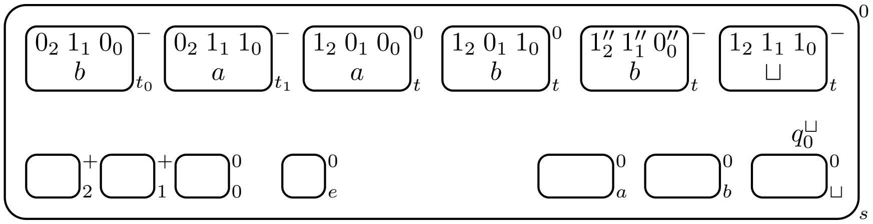

$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1''_2\ 1''_1\ 0''_0$ t $+$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
2 $+$
1 $+$
0 0
e 0
a 0
b 0
$\sqcup\ q_{12}$ $\sqcup$ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1''_2\ 1''_1\ 0''_0$ t $+$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
2 $+$
1 $+$
0 0
e 0
a 0
b 0
$b'\ q_{13}$ $\sqcup$ 0
s 0

The new primed symbol-object is sent back to the only positive tape-membrane as before (see page 47), while the state-object is sent out as a new state-object q_0^γ, having the tape symbol as a superscript and a new counter, starting from 0, as a subscript (there will be no conflict with the previous rules due to the new superscript):

$$\begin{array}{ll} [\sigma']^0_\gamma \to [\ ]^0_\gamma\ \sigma' & \text{for } \gamma, \sigma \in \mathit{\Gamma} \\ [q_{p(n)+11} \to q_{p(n)+12}]^0_\gamma & \text{for } q \in Q \text{ and } \gamma \in \mathit{\Gamma} \\ [q_{p(n)+12}]^0_\gamma \to [\ ]^0_\gamma\ q_0^\gamma & \text{for } q \in Q \text{ and } \gamma \in \mathit{\Gamma}. \end{array}$$


$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1''_2\ 1''_1\ 0''_0$ t $+$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
2 $+$
1 $+$
0 0
e 0
a 0
b 0
b'
q_{14} $\sqcup$ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1''_2\ 1''_1\ 0''_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
2 $+$
1 $+$
0 0
e 0
a 0
b 0
$q_0^\sqcup$
$\sqcup$ 0
s 0

While the doubly-primed bit-objects are reset to their initial state as described earlier, the state-object begins to update the position-membranes, reflecting the movement of the tape head. Recall that incrementing a binary counter means flipping its bits, from the least to the most significant one, until a 0 is flipped into an 1 (i.e., the remaining bits are left unchanged). Similarly, decrementing it means flipping its bits in that order until a 1 is flipped into a 0. The subscript of the state-object, initially 0, records the next bit position to flip. The flipping operation is carried out by entering and exiting the corresponding position-membrane and updating its charge. When a 0 has been flipped into an 1 (for an increment), or a 1 into a 0 (for a decrement), the subscript of the state-object becomes $p(n) + 1$, thus leaving the subsequent bits unchanged.

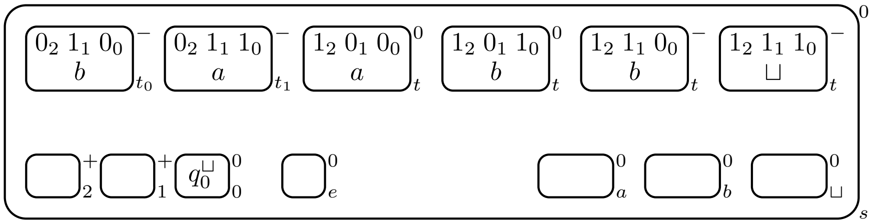
If $\delta(q, \gamma) = (r, \sigma, \triangleright)$ for some $r \in Q$, $\sigma \in \Gamma$ (i.e., the tape head moves to the right), then the position-updating procedure is performed via the following rules:

$$\begin{array}{ll}
q_i^\gamma \; [\;]_i^\alpha \to [q_i^\gamma]_i^\alpha & \text{for } \gamma \in \Gamma,\ q \in Q,\ \alpha \in \{0, +\} \text{ and } 0 \le i \le p(n) \\
[q_i^\gamma]_i^0 \to [\;]_i^+ \; q_{p(n)+1}^\gamma & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n) \\
[q_i^\gamma]_i^+ \to [\;]_i^0 \; q_{i+1}^\gamma & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n)
\end{array}$$

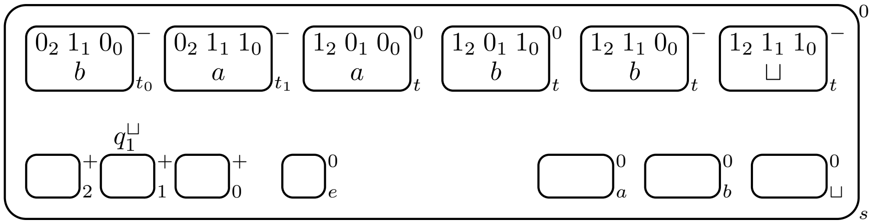
If $\delta(q, \gamma) = (r, \sigma, \triangleleft)$ for some $r \in Q$, $\sigma \in \Gamma$ (i.e., the tape head moves to the left), then the rules are:

$$\begin{array}{ll}
q_i^\gamma \; [\;]_i^\alpha \to [q_i^\gamma]_i^\alpha & \text{for } \gamma \in \Gamma,\ q \in Q,\ \alpha \in \{0, +\} \text{ and } 0 \le i \le p(n) \\
[q_i^\gamma]_i^0 \to [\;]_i^+ \; q_{i+1}^\gamma & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n) \\
[q_i^\gamma]_i^+ \to [\;]_i^0 \; q_{p(n)+1}^\gamma & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n)
\end{array}$$

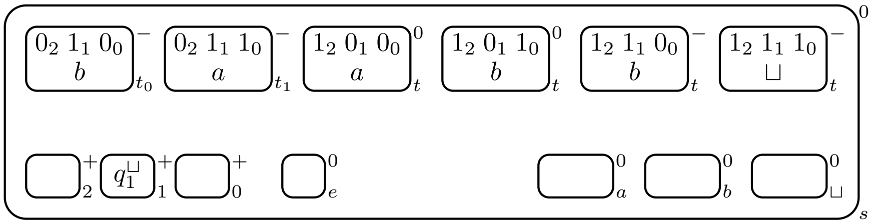
In our example we have to decrement the head position from 110 to 101 by flipping only the two least significant bits (notice how the subscript 2 is skipped):

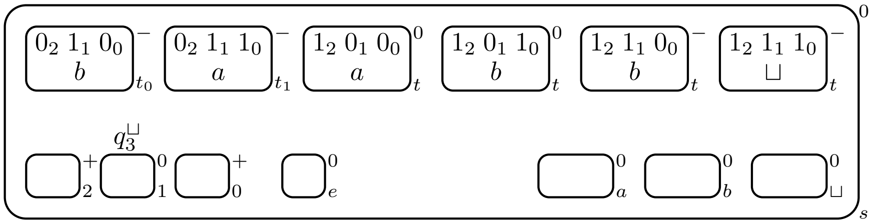


$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
$+$ 2
$+$ 1
$q_0^\sqcup$ 0 0
0 e
0 a
0 b
0 $\sqcup$
0 s



$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
$q_1^\sqcup$
$+$ 2
$+$ 1
$+$ 0
0 e
0 a
0 b
0 $\sqcup$
0 s

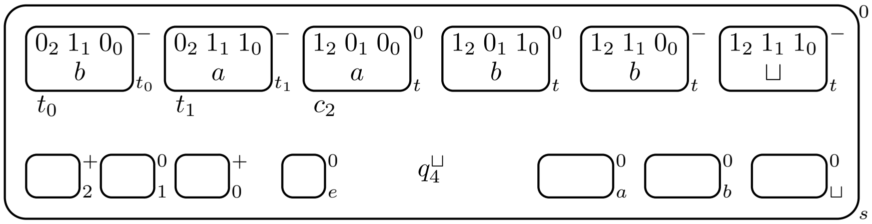

$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
2 $+$
$q_1^{\sqcup}$ 1 $+$
0 $+$
e 0
a 0
b 0
$\sqcup$ 0
s 0

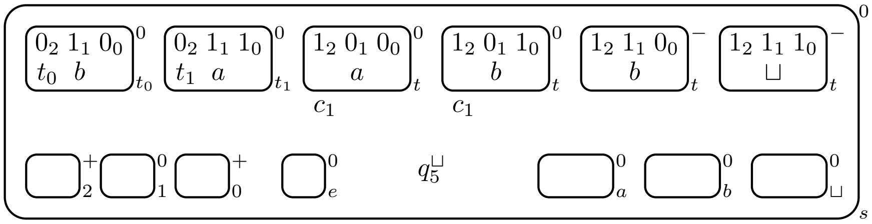

$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
$q_3^{\sqcup}$
2 $+$
1 0
0 $+$
e 0
a 0
b 0
$\sqcup$ 0
s 0

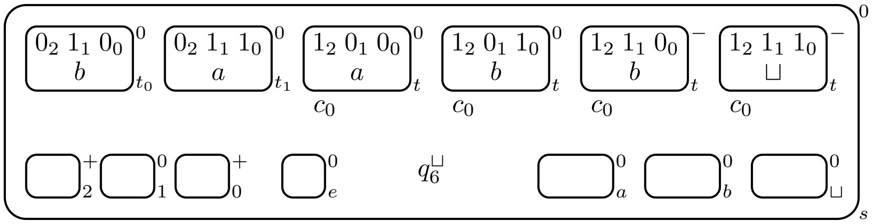
When the subscript of q^γ reaches $p(n)+1$ (i.e., when the position updating has been completed) the system begins preparing the encoding of next configuration of M. This requires resetting all charges of tape-membranes to neutral: this is accomplished by creating n objects $t_0, \dots, t_{n-1}$ (resetting the membranes having the same name) and $2^{p(n)}$ copies of object c_0 (resetting the membranes having label t). The latter objects are created by an initial object $c_{p(n)}$, which is rewritten as two copies of $c_{p(n)-1}$, each of them rewritten as two copies of $c_{p(n)-2}$, and so on. The state-object waits for this process to terminate, and then finally becomes the new state of M, as described by the transition function.

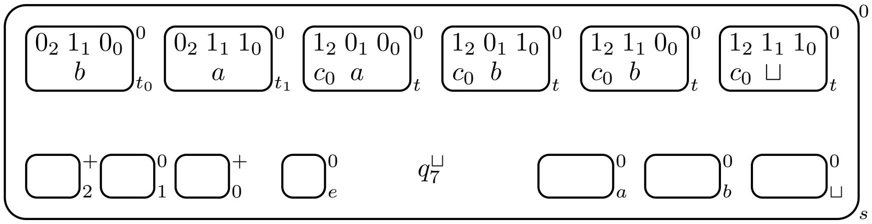
$$\begin{array}{ll}
[q^\gamma_{p(n)+1} \to q^\gamma_{p(n)+2}\ t_0 \cdots t_{n-1}\ c_{p(n)}]^0_s & \text{for } q \in Q \text{ and } \gamma \in \mathit{\Gamma} \\
[q^\gamma_{p(n)+k} \to q^\gamma_{p(n)+k+1}]^0_s & \text{for } q \in Q,\ \gamma \in \mathit{\Gamma} \text{ and } 2 \le k \le p(n)+2 \\
t_j\ [\ ]^\alpha_{t_j} \to [t_j]^0_{t_j} & \text{for } 0 \le j \le n-1 \text{ and } \alpha \in \{0,-\} \\
[t_j \to \lambda]^0_{t_j} & \text{for } 0 \le j \le n-1 \\
[c_i \to c_{i-1}c_{i-1}]^0_s & \text{for } 1 \le i \le p(n) \\
c_0\ [\ ]^\alpha_t \to [c_0]^0_t & \text{for } \alpha \in \{0,-\} \\
[c_0 \to \lambda]^0_t & \\
[q^\gamma_{2p(n)+3} \to r]^0_s & \text{for } q, r \in Q \text{ and } \gamma \in \mathit{\Gamma}, \text{ if } \delta(q,\gamma) = (r,\sigma,d) \\
 & \text{for some } \sigma \in \mathit{\Gamma} \text{ and } d \in \{\triangleleft, \triangleright\}.
\end{array}$$

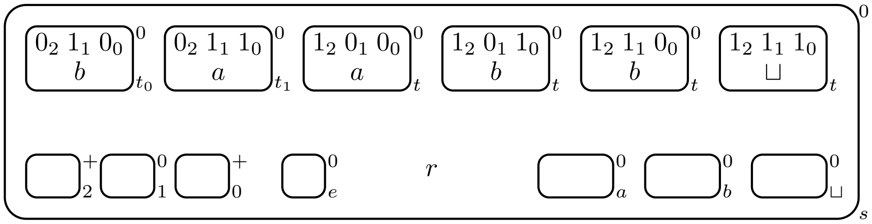
In our example, the computation evolves as follows:


$0_2\ 1_1\ 0_0$ b t_0 $-$
$0_2\ 1_1\ 1_0$ a t_1 $-$
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
t_0 t_1 c_2
2 $+$ 1 0 0 $+$ e 0
$q_4^{\sqcup}$
a 0 b 0 $\sqcup$ 0
s 0


$0_2\ 1_1\ 0_0$ $t_0\ b$ t_0 0
$0_2\ 1_1\ 1_0$ $t_1\ a$ t_1 0
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
c_1 c_1
2 $+$ 1 0 0 $+$ e 0
$q_5^{\sqcup}$
a 0 b 0 $\sqcup$ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0_2\ 1_1\ 1_0$ a t_1 0
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t $-$
$1_2\ 1_1\ 1_0$ $\sqcup$ t $-$
c_0 c_0 c_0 c_0
2 $+$ 1 0 0 $+$ e 0
$q_6^{\sqcup}$
a 0 b 0 $\sqcup$ 0
s 0

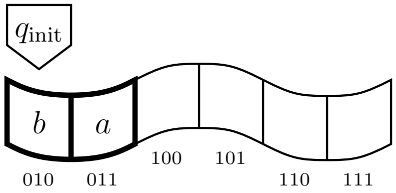

$0_2\ 1_1\ 0_0$ b t_0 0
$0_2\ 1_1\ 1_0$ a t_1 0
$1_2\ 0_1\ 0_0$ $c_0\ a$ t 0
$1_2\ 0_1\ 1_0$ $c_0\ b$ t 0
$1_2\ 1_1\ 0_0$ $c_0\ b$ t 0
$1_2\ 1_1\ 1_0$ $c_0\ \sqcup$ t 0
2 $+$ 1 0 0 $+$ e 0
$q_7^{\sqcup}$
a 0 b 0 $\sqcup$ 0
s 0


$0_2\ 1_1\ 0_0$ b t_0 0
$0_2\ 1_1\ 1_0$ a t_1 0
$1_2\ 0_1\ 0_0$ a t 0
$1_2\ 0_1\ 1_0$ b t 0
$1_2\ 1_1\ 0_0$ b t 0
$1_2\ 1_1\ 1_0$ $\sqcup$ t
2 $+$ 1 0 0 $+$ e 0
r
a 0 b 0 $\sqcup$ 0
s 0

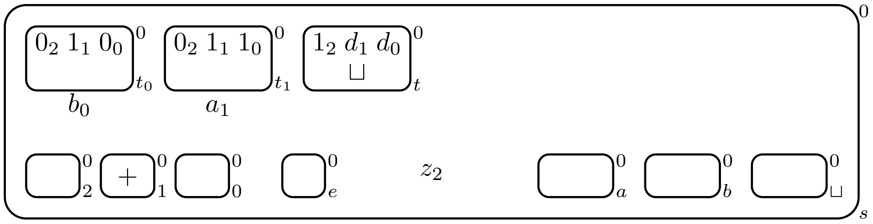
We have finally reached the configuration of Π_x corresponding to the configuration of M after it has performed its computation step, and we are ready to start simulating a new step of M.

3.3 Creating the initial configuration

In the previous section we described how to simulate a computation step of M starting from an arbitrary configuration of the Turing machine. We still need to describe how to encode the *initial* configuration of M (represented in the following picture) in the P system Π_x simulating it.



We use the following as the initial configuration *of the P system*:



This initial configuration consists of a membrane s containing:

- Membranes $t_0, \ldots, t_{n-1}$, each containing $p(n) + 1$ bit-objects encoding the position numbers of the input cells (as described above).
- One single copy of membrane t, containing the bit-object $1_{p(n)}$ (recall that the most significant bit for the non-input tape cells is always 1) and the "bit variables" $d_0, \ldots, d_{p(n)-1}$.
- The position-membranes, labeled by $0, \ldots, p(n) + 1$, whose initial charge is 0 by definition. Those which have to be set to positive in order to set up the initial head position (i.e., $2^{p(n)} - n$) contain a $+$ object.
- The error-membrane e.
- The symbol-membranes, labeled by the elements of Γ.
- The object $z_{p(n)}$.

All these items only depend on the *size* of the input of the Turing machine M. The input itself is encoded by a set of objects denoting the symbols, subscripted by an index indicating their position in the string (counting from 0), and placed into the input membrane s. In our example, the input ba of M is encoded in Π_x as b_0a_1.

During the initialization phase of Π_x, several operations are carried out. First of all, the input-objects are sent to the corresponding tape-membranes (indicated in their subscripts). This is accomplished by using the following rules:

$$\gamma_i\ [\ ]_{t_i}^0 \to [\gamma]_{t_i}^0 \qquad \text{for } \gamma \in \Gamma \text{ and } 0 \le i \le n-1.$$

The position-membranes have their charges set to + by sending out the + objects, which are then deleted using

$$\begin{aligned} &[+]_i^0 \to [\ ]_i^+\ + && \text{for } 0 \le i \le p(n) \\ &[+ \to \lambda]_s^0. \end{aligned}$$

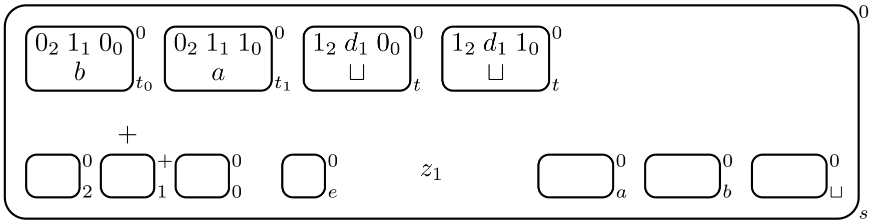
The tape-membranes corresponding to the working portion of the tape, of size $2^{p(n)}$, are created by iterated elementary membrane division, starting from the single initial tape-membrane t. The objects d_i are rewritten as 0_i on one side, and as 1_i on the other, whenever the membrane is divided. This process creates all the $2^{p(n)}$ cell numbers in binary. The corresponding rules are

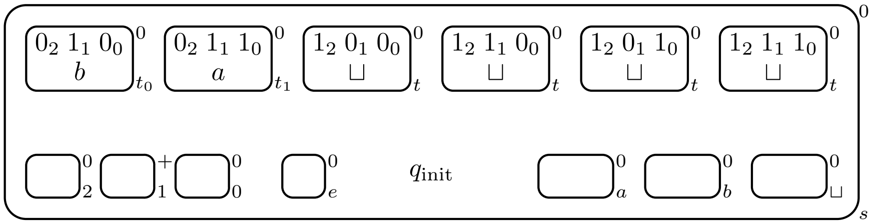
$$[d_i]_t^0 \to [0_i]_t^0\ [1_i]_t^0 \qquad \text{for } 0 \le i \le p(n)-1.$$

This latter operation requires $p(n)$ steps. The object $z_{p(n)}$ has its subscript decremented to 1, and then finally becomes the state-object corresponding to the initial state of M:

$$\begin{aligned} &[z_i \to z_{i-1}]_s^0 && \text{for } 2 \le i \le p(n) \\ &[z_1 \to q_{\text{INIT}}]_s^0. \end{aligned}$$

In our example, the initialization phase proceeds as follows.





After having initialized the P system Π_x according to the initial configuration of M on input x, the simulation is carried out step-by-step as described in the previous section.

3.4 Halting and output

The only missing part of our simulation concerns the operations to carry out when the simulated machine halts by accepting or rejecting. Assuming the transition function δ of M is undefined on its accepting state q_{YES} and its rejecting state q_{NO}, we can simply proceed as follows: if the machine enters q_{YES}, then in Π_x the state-object q_{YES} appears inside the outermost membrane s; we can then send that object out to the environment as YES to make Π_x accept. The behavior is analogous for the rejecting state q_{NO}.

$$[q_{\text{YES}}]_s^0 \to [\,]_s^0 \text{ YES} \qquad\qquad [q_{\text{NO}}]_s^0 \to [\,]_s^0 \text{ NO}.$$

3.5 Completing the proof

The initialization phase of Π_x, simulating M on an input x of length n, requires $O(p(n))$ time in order to create $2^{p(n)}$ copies of membrane t by a sequence of elementary divisions.

Then, the $t(n)$ steps performed by M are simulated. Each step involves guessing the tape-membrane corresponding to the cell currently under the tape head; each simulated step may require up to $O(s(n))$ guesses in the worst case. For each guess, we need to check if the correct tape-membrane was selected, and this requires time proportional to the number of bit positions, i.e., $O\big(\log s(n)\big)$ steps. If the membrane is incorrect, then $O(1)$ steps are required to set its charge to negative and prepare the system for a further guess. If the membrane was the right one, the state, head position and tape symbol have to be updated, and this requires further $O\big(\log s(n)\big)$ steps. Hence, each simulated step of M requires $O\big(s(n)\log s(n)\big)$ steps of Π_x, for a total of $O\big(t(n)s(n)\log s(n)\big)$ steps.

As the output step only requires constant time, the whole simulation can be carried out in $O\big(t(n)s(n)\log s(n)\big)$ time. Since $s(n)$ is $O(t(n))$ for a Turing machine (assuming it at least reads its whole input), the simulation time can be expressed as a function of $t(n)$ as $O\big(t(n)^2\log t(n)\big)$. Hence, this is an "efficient" simulation: if M works in polynomial time, then the family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ simulating it also works in polynomial time; if M runs in exponential (resp., doubly-exponential) time, then $\boldsymbol{\Pi}$ also runs in exponential (resp., doubly-exponential time).

Notice that the actual running time of the simulation depends on the sequence of nondeterministic choices performed when the system has to guess the correct tape-membrane. In the best case, when the correct guess is always the first one, the time reduces to $O\big(t(n)\log s(n)\big)$ instead of $O\big(t(n)s(n)\log s(n)\big)$ as in the worst case.

The space required by Π_x is asymptotically due to the tape-membranes. These are $s(n)$ in number, and each of them contains $O(\log s(n))$ bit-objects denoting its position on the tape. Hence, the simulation requires $O\big(s(n)\log s(n)\big)$ space: a polynomial-space Turing machine is simulated in polynomial space, and an exponential-space one in exponential space.

In order to complete the proof of Theorem 1, we only need to check that the family $\boldsymbol{\Pi}$ is polynomial-time uniform. It is easy to verify that all the rules and the initial configuration of Π_x actually depend only on the length of x (except for the input objects). There is a constant number of different kinds of rules parametric with respect to n or $p(n)$; the larger sets of rules are (1) and (2) on page 47, consisting of $O\big(n \times p(n)\big)$ rules each.

4 Characterizing exponential space

In the previous section we described a simulation of deterministic Turing machines working in exponential space by means of P systems. Combining this result with the converse simulation illustrated in [8], we can show that the computational power of Turing machines and of P systems with active membranes coincide when these devices operate within an exponential space limit:

Corollary 1. *The following inclusions hold:*

$$\begin{array}{ccc}
\mathbf{EXPMCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} & \subseteq & \mathbf{EXPMCSPACE}^{[\star]}_{\mathcal{AM}} \\
\cup\!\mid & & \mid\!\cap \\
\mathbf{EXPSPACE} & \supseteq & \mathbf{NEXPMCSPACE}^{\star}_{\mathcal{AM}}
\end{array}$$

where $[\star]$ *denotes optional semi-uniformity (instead of uniformity). Hence, all classes shown in the diagram coincide.*

Proof. The chain of inclusions

$$\mathbf{EXPMCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} \subseteq \mathbf{EXPMCSPACE}^{[\star]}_{\mathcal{AM}} \subseteq \mathbf{NEXPMCSPACE}^{\star}_{\mathcal{AM}}$$

holds by definition. That $\mathbf{NEXPMCSPACE}^{\star}_{\mathcal{AM}} \subseteq \mathbf{EXPSPACE}$ is an immediate corollary of Theorem 5 in [8]. Finally, the inclusion of $\mathbf{EXPSPACE}$ in $\mathbf{EXPMCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ directly follows from Theorem 1. □

Let us remark that the power of the complexity class $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$ is mostly due to the families of P systems themselves, as opposed to the Turing machines providing the uniformity condition; indeed, these would only be able to solve the strictly smaller [4] class $\mathbf{P}$ of decision problems.

5 Conclusions

We showed that the class of problems solvable by P systems with active membranes in exponential space coincides with the class of problems solved by Turing machines in exponential space, that is, $\mathbf{EXPMCSPACE}_{\mathcal{AM}} = \mathbf{EXPSPACE}$.

Again, the techniques used to prove this result cannot be applied immediately when the space bound is less strict, i.e., super-exponential. In fact, in this case we would need systems using indexed bits, where the index ranges over a super-polynomial set of values; as a consequence, such systems cannot be generated in a uniform way in a polynomial number of steps, as requested by Definition 3. Thus, it remains open for this case the question whether these kinds of P systems with active membranes have the same computing power as Turing machines working under the same space constraints.

Let us note that if membrane creation [1] is used instead of membrane division, then the simulation may be straightforward and faster (the slowdown would be by a constant factor only). The simulation would also be deterministic, instead of requiring "wild" nondeterminism as in our result. Turing machine cells may be represented by *nested* membranes, created when needed; this is a construction that generalizes even to super-exponential space. However, with membrane division only, the depth of membrane hierarchy cannot increase during the computation, and it is originally polynomial under our current definition.

As a direction for future research, it might also be interesting to analyze the behavior of families of P systems with active membranes working in logarithmic space. However, there are two major issues to be considered in this case: first, we should slightly change the notion of space complexity, in order to allow for a "read-only" input multiset that is not counted when the space required by the P system is measured (similarly to the input tape of a logspace Turing machine). Furthermore, the notion of uniformity used to define the families of P systems should be weakened, since polynomial-time Turing machines constructing the families might be able to solve the problems altogether by themselves. More general forms of uniformity have already been investigated [3], and that work is going to be useful when attacking this problem.

Acknowledgements

Artiom Alhazov gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618). The work of the other authors was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

References

1. Artiom Alhazov, Rudolf Freund, and Agustín Riscos-Núñez. Membrane division, restricted membrane creation and object complexity in P systems. *International Journal of Computer Mathematics*, 83(7):529–547, 2006.
2. Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.

3. Niall Murphy and Damien Woods. The computational power of membrane systems under tight uniformity conditions. *Natural Computing*, 10(1):613–632, 2011.
4. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.
5. Mario J. Pérez-Jiménez, Álvaro Romero-Jiménez, and Fernando Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–284, 2003.
6. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. Introducing a space complexity measure for P systems. *International Journal of Computers, Communications & Control*, 4(3):301–310, 2009.
7. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems simulating oracle computations. In Marian Gheorghe, Gheorghe Păun, Arto Salomaa, Grzegorz Rozenberg, and Sergey Verlan, editors, *Membrane Computing, 12th International Conference, CMC 2011*, volume 7184 of *Lecture Notes in Computer Science*, pages 346–358. Springer, 2011.
8. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science*, 22(1):65–73, 2011.
9. Gheorghe Păun. P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
10. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
11. Petr Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, 2003.
12. Claudio Zandron, Claudio Ferretti, and Giancarlo Mauri. Solving NP-complete problems using P systems with active membranes. In Ioannis Antoniou, Cristian S. Calude, and Michael J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference*, pages 289–301. Springer, 2001.

The Power of Symport-3 with Few Extra Symbols

Artiom Alhazov[1,2] and Yurii Rogozhin[2]

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
E-mail: `artiom.alhazov@unimib.it`

[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: {`artiom`,`rogozhin`}`@math.md`

Summary. Membrane systems (with symbol objects) are formal models of distributed parallel multiset processing. Symport rules move multiple objects to a neighboring region. It is known that P systems with symport rules of weight at most 3 and a single membrane are computationally complete with 7 superfluous symbols. It is also known that without any superfluous symbols such systems only generate finite sets.

We improve the lower bounds on the generative power of P systems with few superfluous objects as follows. 0: empty set and all singletons; k: all sets with at most k elements and all sets of numbers k+regular with up to k states, $1 \leq k \leq 5$; 6: all regular sets of non-negative integers. All results except the last one are also valid for different modes, e.g., sequential one, also for higher values of k.

1 Introduction

Membrane systems (with symbol objects) are formal models of distributed parallel multiset processing. Symport rules move predefined groups objects to a neighboring region [4]. In maximally parallel mode (typical for membrane computing), this alone is sufficient to construct a computationally universal device, as long as the environment may contain an unbounded supply of some objects. The number of symbols specified in a symport rule is called its weight. The result of a computation is the total number of objects when the system halts. In some cases, however, for technical reasons the desired result may only be obtained alongside a small number of superfluous objects in the output region.

There were multiple papers improving the results on P systems with symport/antiport of small weight (an antiport rule moves objects between 2 regions in both directions, and its weight is the maximum of objects per direction), see [2] for a survey of results. Computational completeness is achieved even for minimal cooperation: either symport/antiport of weight 1, or symport of weight at most

2. This holds for 2 membranes, without superfluous objects if the output is considered in the skin, or with 1 superfluous object under the classical assumption of the output in the elementary membrane. In the tissue case, the accepting systems can even be made deterministic.

With cooperation of up to 3 objects, a single membrane suffices. The regions are called the skin and the environment, the latter contains an unbounded supply of some objects, while the contents of the former is always finite. With antiport-2/1 alone (i.e., exchanging 1 object against 2), the computational completeness is obtained with a single superfluous object. With symport-3 (i.e., symport rules only, of weight up to 3), one proved in [3] that 13 extra objects suffices for computational completeness. This result has been improved in [1] to 7 superfluous symbols. In the same paper it was shown that without any superfluous symbols such systems only generate finite sets.

The computation consists of multiple, sometimes simultaneous, actions of two types: move objects from the skin to the environment, and move objects from the environment into the skin. It is obvious that trying to move all objects out in the environment will activate the rules of the second type. Since, clearly, rules of the first type alone cannot generate more than finite sets, it immediately follows that the "garbage" is unavoidable. This paper tries to improve the currently best bounds on how much "garbage" is sufficient.

2 Definitions

Throughout the paper, by "number" we will mean a non-negative integer. We write N_jFIN_k to denote the family of all sets of numbers each not smaller than j, of cardinality k. By N_jREG_k we denote the family of all sets M of numbers each not smaller than j, such that $\{x-j \mid x \in M\}$ is accepted by some finite automaton with k states, with at least one transition from every non-final state. We assume the reader to be familiar with the basics of the formal language theory, and we recall that for a finite set V, the set of words over V is denoted by V^*, the set of non-empty words is denoted by V^+, and a multiset may be represented by a string, representing the multiplicity by the number of occurrences, the order not being important.

2.1 Finite Automata

Definition 1. *A finite automaton is a tuple $A = (\Sigma, Q, q_0, \delta, F)$, where Σ is an input alphabet, Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \longrightarrow 2^Q$ is the transition mapping.*

The function δ is naturally extended from symbols to strings. The language accepted by A is the set $\{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$.

2.2 P Systems with Symport

The scope of this paper is limited to P systems with symport only, with a single membrane.

Definition 2. *A P system with symport rules and one membrane is a tuple*

$$\Pi = (O, E, \mu = [\ ]_1, w, R), \text{ where}$$

- *O is a finite set called alphabet; its elements are called symbols,*
- *$E \subseteq O$ is the set of objects appearing in the environment in an unbounded supply,*
- *μ is the membrane structure, trivial in case of one membrane; the inner region is called the skin and the outer region is called the environment,*
- *$w \in O^*$ is the specification of the initial contents of the inner region,*
- *R is the set of rules of types (u, out) or (v, in), $u, v \in O^+$.*

An action of a rule (u, out) is to move the multiset of objects specified by u from the skin into the environment. An action of a rule (v, in) is to move the multiset of objects specified by v from the environment into the skin ($v \in E^*$ is not allowed by definition). The objects are assigned to rules non-deterministically. Maximal parallelism allows application of multiple rules simultaneously and multiple times, as long as there are enough copies of objects for them, and provided no further rule is applicable to the unassigned objects.

The computation halts when no rules are applicable at some step. The result of a halting computation is the total number of objects in the inner region when it halts. The set of numbers $N(\Pi)$ generated by a P system Π is the set of results of all its computations.

The family of sets of numbers generated by a family of P systems with one membrane and symport rules of weight at most k is denoted by $NOP_1(sym_k)$ in maximally parallel mode. We add superscript *sequ* to P to indicate sequential mode instead.

3 Few-Element Sets

We now present a few simple systems.

$$\Pi_0 = (O = \{a\}, E = \emptyset, \mu = [\ ]_1, w = a, R = \{(a, in), (a, out)\}).$$

System Π_0 perpetually moves a single object in and out, effectively generating the emptyset. For any $x \in \mathbb{N}$, setting $R = \emptyset$ and $w = a^x$ will lead to a system Π_1 which immediately halts, generating a singleton $\{x\}$.

We now proceed to arbitrary small-cardinality sets. To generate a multi-element set, the system must make at least one non-deterministic choice. Since we want to allow the difference between the elements to be arbitrarily large, such choice must be persistent, i.e., the decision information should not vanish, at least until multiple objects are moved accordingly. For any numbers $y > x$, consider the following P system:

$$\begin{aligned}\varPi_2 &= (O = \{a,b,i,p,q\}, E = \{q\}, \mu = [\ ]_1, w = a^x b^{y-x+1} ip, R),\\ R &= \{(i, out), (ip, out), (pq, in), (pqb, out)\}.\end{aligned}$$

There are two possible computations of $\varPi_2$: either i exits alone, halting with $a^x b^{y-x+1} p$, generating $y+2$, or i exits with p, leading to a sequence of application of the last two rules until no objects b remain in the skin, halting with $a^x pq$, generating $x+2$. Therefore, $\varPi_2$ generates an arbitrary 2-element set with 2 extra objects.

This construction can be improved to generate higher-cardinality sets as follows. Let $m \geq 2$; for arbitrary $m+1$ distinct numbers denote the largest one by y and the others by x_j, $1 \leq j \leq m$. We construct another P system:

$$\begin{aligned}\varPi_{m+1} &= (O, E = \{q_j \mid 1 \leq j \leq m\}, \mu = [\ ]_1, w, R),\\ O &= \{a_j \mid 1 \leq j \leq y+1\} \cup \{i\} \cup \{p_j, q_j \mid 1 \leq j \leq m\},\\ w &= i \prod_{j=1}^{y+1} a_j \prod_{j=1}^{m} p_j,\\ R &= \{(i, out)\} \cup \{(ip_j, out), (p_j q_j, in) \mid 1 \leq j \leq m\}\\ &\quad \cup (p_j q_j a_k, out) \mid 1 \leq j \leq m,\ x_j + 1 \leq k \leq y+1, j \neq k\}.\end{aligned}$$

Such system behaves like $\varPi_2$, except it also chooses among different objects p_j to send out symbols a_k for $k > x_j$. It halts either with $a_1 \cdots a_{y+1} p_1 \cdots p_m$ generating $y+m+1$, or with $a_1 \cdots a_{x_j} q_j p_1 \cdots p_m$ generating $x_j + m + 1$. For $m = 2, 3, 4$ this leads to $\varPi_3$ generating $\{x_1+3, x_2+3, y+3\}$, $\varPi_4$ generating $\{x_1+4, x_2+4, x_3+4, y+4\}$, and $\varPi_5$ generating $\{x_1+5, x_2+5, x_3+5, x_4+5, y+5\}$, i.e., any 3-, 4- or 5-element set with 3, 4 or 5 extra objects, respectively.

4 Sequential Mode and Straightforward Regularity

Remark 1. All P systems constructed in the previous section de facto work sequentially, hence the results are valid for P system in any other traditional derivation mode (e.g., sequential, asynchronous, minimally parallel, maximal strategy).

The sequential mode is an interesting candidate for a research topic, due to its simplicity, even though its power (as that of any sequential multiset rewriting system without control) cannot exceed $NMAT = NREG$, and infinite sets without zero cannot be generated in this case either, for the same argument as in maximally parallel case.

We now proceed by constructing a P system generating the length set of a language accepted by a finite automaton $A = (Q, \varSigma, \delta, q_0, F)$, where $Q = \{q_j \mid$

$0 \le j \le m\}$; we assume A satisfies the following property: there is at least one transition from every non-final state.

$$\begin{aligned}\Pi_A = (O = Q \cup Q' \cup \Sigma, E = Q' \cup \Sigma, \mu = [\ ]_1, w = q_0 \cdots q_m q_0', R),\\ R = \{(q_j q_j', out) \mid 0 \le j \le m\}\\ \cup \{(q_j a q_k', in) \mid q_k \in \delta(q_j, a),\ a \in \Sigma\} \cup \{(q_j', out) \mid q_j \in F\}.\end{aligned}$$

Notice also that adding an arbitrary number of objects from Σ to the initial configuration increases the result by the corresponding number. Unfortunately, besides the needed number, the skin region at halting also contains the superfluous symbols, as many as there are states in A. Therefore, we have obtained all sets N_jREG_k, $j \ge k$. Within this section it is enough to consider $j = k$, because REG_j contains REG_k.

Remark 2. If we fix j, k and restrict A to be deterministic, then the number set M generated by the corresponding P system can be characterized by the following properties: $x \ge j$ for all $x \in M$ and there exists a number $0 \le p \le k$ such that if $x \ge j + k - p$, then $x \in M$ if and only if $x + p \in M$. Hence, acceptability of sufficiently large numbers is determined by the remainder of their division by some $p < k$. The general result is also valid for non-deterministic systems, but exact characterization in terms of states is less straightforward.

The simplest examples of application of Π_A are the set of all positive numbers and the set of all positive even numbers.

It is not difficult to notice that the result of Π_A does not depend on the computation mode, e.g., it is valid also for maximally parallel mode. Therefore, both $NOP_1^{sequ}(sym_3)$ and $NOP_1(sym_3)$ contain

$$NFIN_0 \cup NFIN_1 \cup \bigcup_{k=2}^{\infty} N_kFIN_k \cup \bigcup_{k=1}^{\infty} N_kREG_k.$$

We recall that the upper bound for $NOP_1^{sequ}(sym_3)$ is $N_1REG \cup NFIN$.

5 Larger Sets and Few Extra Objects

We now focus on the maximally parallel mode, and revisit the symport-3 construction from [1]. The 7 extra objects were denoted $l_h, b, d, x_1, x_4, x_5, x_6$.

Recall that any regular set of numbers is generated by some non-deterministic register machine with only ADD-instructions, or, equivalently, accepted by a finite automaton. It suffices to take the same construction, and notice that object d is no longer needed to check for the conflicting counters. Removing it from the system leads to P systems generating N_6REG. Hence, the contribution of the previous section for maximally parallel P systems can be limited to infinite sets of the form small number+infinite regularity accepted by at most 5 states.

We now recall the construction from [1] with the corresponding changes to generate N_6REG, rewritten in the syntax of finite automata. Let L be an arbitrary set from N_6REG. Then there exists a finite automaton $A = (\Sigma, Q, q_0, \delta, F)$ accepting $L - 6 = \{n - 6 \mid n \in L\}$. We construct a P system simulating A:

$$\begin{aligned}
&\Pi = (O, E, \ [\ \ ]_1\ , w, R, 1), \text{ where}\\
&O = \{x_i \mid 1 \leq i \leq 6\} \cup Q \cup \{(p, q, j) \mid p, q \in Q,\ 1 \leq j \leq 3\} \cup \{a, A, b, H\},\\
&E = Q \cup \{(p, q, 2) \mid p, q \in Q\} \cup \{a, A, x_2, x_3, H\},\\
&w = q_0 x_1 x_4 x_5 x_6 bA \prod_{p,q \in P} (p, q, 1)(p, q, 3),\\
&R = \{1 : (x_1x_2x_3, in),\ 2 : (x_2x_4x_5, out),\ 3 : (x_3x_6, out),\ 4 : (Hb, in)\}\\
&\quad \cup \{5 : (qb, out) \mid q \in F\}\\
&\quad \cup \{6 : (Hbx, out) \mid x \in \{(p, q, 1), (p, q, 3) \mid p, q \in Q\} \cup \{A\}\}\\
&\quad \cup \{7 : (p(p, q, 1)x_1, out),\ 8 : ((p, q, 1)x_4(p, q, 2), in),\\
&\qquad 9 : ((p, q, 2)(p, q, 3)A, out), 10 : ((p, q, 3)x_5q, in),\\
&\qquad 11 : (Ax_6a, in) \mid q \in \delta(p, s)\}.
\end{aligned}$$

Notice that the effect of rules 1,2,3 is that sending object x_1 out will bring x_2 and x_3 in, which, in turn, will send objects x_4, x_5 and x_6 out. Notice also that rules 8,10,11 need x_4, x_5 and x_6, so if these objects are inside the membrane, then all objects of the form (p, q, j), $j \in \{1, 3\}$ may be sent out, without enabling these rules. We will skip mentioning objects x_j in the simulation.

The simulation of a transition in A is performed as follows:

- The state p brings object $(p, q, 1)$ out, also sending x_1 out to enable the rest of the simulation.
- Object $(p, q, 1)$ brings object $(p, q, 2)$ in, which, in turn, brings both $(p, q, 3)$ and A out.
- Object $(p, q, 3)$ brings in the next state q, while object A brings in object a, contributing to the result.

If the current state is final, rule 5 may be applied, leading to iteration of rules 4 and 6, taking out all objects except H, b, x_1, x_4, x_5, x_6 and the desired number of copies of a.

Remark 3. If we were interested in generating vectors rather than numbers, simulation of partially blind register machines could be also performed with six additional objects. Since this gives no additional power for numbers (i.e., NMAT=NREG), we presented the simpler construction, without subtraction.

6 Discussion

It has been known that P systems with symport rules of weight at most 3 generate at least N_7RE, and cannot generate infinite sets containing 0. We have improved the lower bound to

$$\begin{gathered}NFIN_0 \cup NFIN_1 \cup N_2FIN_2 \cup N_3FIN_3 \cup N_4FIN_4 \cup N_5FIN_5\\ \cup N_1REG_1 \cup N_2REG_2 \cup N_3REG_3 \cup N_4REG_4 \cup N_5FIN_5\\ \cup N_6REG \cup N_7RE.\end{gathered}$$

It is open whether this bound is tight, since the current best known upper bound is $N_1RE \cup NFIN$.

For the sequential case, the bounds are given in the end of Section 4. It is particularly interesting whether infinitely many additional objects are unavoidable for generation of regular number sets in the sequential mode.

Acknowledgements

The first author acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618).

References

1. A. Alhazov, R. Freund, Yu. Rogozhin: Computational Power of Symport/Antiport: History, Advances and Open Problems. In: R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa: *Membrane Computing, 6th International Workshop*, WMC 2005, Vienna, Revised Selected and Invited Papers. *Lecture Notes in Computer Science* **3850**, Springer, 2006, 1–30.
2. R. Freund, A. Alhazov, Yu. Rogozhin, S. Verlan: Communication P Systems. Chapter 5 in: Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, 2010, 118–143.
3. P. Frisco, H.J. Hoogeboom: P Systems with Symport/Antiport Simulating Counter Automata. *Acta Informatica* **41**, 2004, 145-170.
4. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing* **20**, 2002, 295-305.

Counting Cells with Tissue-like P Systems

Ioan Ardelean[1,2], Daniel Díaz-Pernil[3], Miguel A. Gutiérrez-Naranjo[4], Francisco Peña-Cantillana[4], Raúl Reina-Molina[3], Iris Sarchizian[1]

[1]Ovidius University, Constanţa 90052, Romania
[2]Institute of Biology Bucharest, Romanian Academy, 060031 Bucharest, Romania
ioan.ardelean57@yahoo.com, irissarchizian@yahoo.com

[3]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics - University of Sevilla, 41012, Spain
sbdani@us.es, raulrm75@gmail.com

[4]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
magutier@us.es, frapencan@gmail.com

Summary. Counting the number of cells obtained in an experiment is crucial in many areas in Biology. Nonetheless, this is usually performed by hand by the researcher due the intrinsic difficulty of the task. In this paper, we present a set of techniques for counting cells inspired in the treatment of Digital Images via tissue-like P systems with promoters.

1 Introduction

Due to the increasing amount of information stored as visual data, the development of new software for dealing efficiently with digital images becomes a necessity. The number of application areas is growing and the progress of new technology needs the design of new software for handling such information. Among the classical areas, we can cite biometrics [1], surveillance [15] or medical imaging [3], but there are many others.

Recently, a new research line has been open by applying well-known membrane computing techniques for solving problems from digital imagery. For example, the *segmentation* problem, [13, 14, 16, 17, 39], *thresholding* [12] or *smoothing* [33]. Special attention deserves [20], where the *symmetric dynamic programming stereo* (SDPS) algorithm [21] for stereo matching was implemented by using simple P modules with duplex channels.

We focus here on a problem from Microbiology. Automated image analysis is increasingly used in Microbiology to quantify important parameters for research and application. The most studied so far are the following: cell numbers, cell volumes, frequencies of dividing cells, in situ classification of bacteria, enumeration of actively respiring bacteria, characterization of bacterial growth on solid medium, viability and physiological activity in biofilms (e.g. [4, 5, 19, 23, 26, 30, 11, 34, 35, 37, 38, 40, 41, 43]). In this paper we report our study of the application of Membrane Computing techniques to the problem of counting cells and show some preliminary results. The whole process is a combination of different techniques of processing images (binarization, segmentation, noise reduction ...) which can be performed by different families of P systems. The final algorithm is a sequence of partial processes which can be performed by Membrane Computing techniques, and the application of such processes can be seen as a global machine which takes as input a digital image showing a biological entity (usually, a photograph taken with a microscopy in a wet lab) and the output is the number of cells in the picture.

The different families of P systems used in the stages of the process have inspired parallel software programs which have been developed by using a device architecture called CUDA™, (Compute Unified Device Architecture). CUDA™ is a general purpose parallel computing architecture that allows the parallel NVIDIA[1] Graphics Processors Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU. GPUs constitute nowadays a solid alternative for high performance computing, and the advent of CUDA allows programmers a friendly model to accelerate a broad range of applications. This novel architecture has been previously used to implement parallel software that simulates the behavior of P systems [6, 8, 9, 10, 32, 33], and, in a similar way than in other implementations, the obtained results in the problem of counting cells are quite promising.

The paper is organized as follows: Next, we recall the computational model used to design the different families of P systems that performs the stages of the algorithm. In section 3, we outline the steps of the process that takes as an input a digital image taken in a wet lab and outputs the number of cells in the image. Section 4 shows an illustrative example and some details of the CUDA implementation. The paper finishes with some conclusions and open lines for a future research.

2 Formal Framework

Next, we recall some basics on the P system model chosen for implementing the solution described below. The model is tissue-like P systems with promoters. Promoters are usually defined on cell-like models [24] and its extension to tissue-like is quite natural. Next, we recall the formal definition.

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

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

where

[1] http://www.nvidia.com.

1. *Γ is a finite alphabet, whose symbols will be called objects;*
2. *$\Sigma \subseteq \Gamma$ is the input alphabet;*
3. *$\mathcal{E} \subseteq \Gamma$ is a finite alphabet representing the set of the objects in the environment available in an arbitrary large amount of copies;*
4. *$w_1, \dots, w_q$ are strings over Γ representing the multisets of objects associated with the cells in the initial configuration;*
5. *$\mathcal{R}$ is a finite set of rules of the following form:*

$$(pro \,|\, i, u/v, j), \quad \textit{for } 0 \leq i \neq j \leq q, \ pro, u, v \in \Gamma^*$$

In these rules, the labels $1, \dots, q$ correspond to the q cells and the label 0 corresponds to the environment;

6. *$i_{in} \in \{1, 2, \dots, q\}$ denotes the input region;*
7. *$i_{out} \in \{1, 2, \dots, q\}$ denotes the output region.*

The rule $(pro \,|\, i, u/v, j)$ 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 i the objects of the promoter *pro* are present. The promoter is not modified by the application of the rule. If the promoter is empty, we will write $(i, u/v, j)$ instead of $(\emptyset \,|\, i, u/v, j)$.

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. A *configuration* is an instantaneous description of the system Π, and it is represented as a tuple $(w_0, w_1, \dots, w_q)$, where $w_1, \dots, w_q$, where represent the multiset of objects contained in the q cells and w_0 represent the multiset of objects from $\Gamma - \mathcal{E}$ placed in the environment (initially $w_0 = \emptyset$). 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.

3 Counting Cells

Counting cells in a picture taken by a microscopy in a wet lab is a hard task. The study of the cells is usually made in such conditions (light filters, noise data, aqueous media, ...) where it is difficult for the human expert to decide whether a spot in the image correspond to a cell or not. In such conditions, the development of a software that provides the exact number of spots in the image that correspond to cells is impossible, since two different experts hardly agree in this issue.

From this starting point, and bearing in mind that the research in Microbiology needs computer aid, we propose a Membrane Computing protocol for obtaining

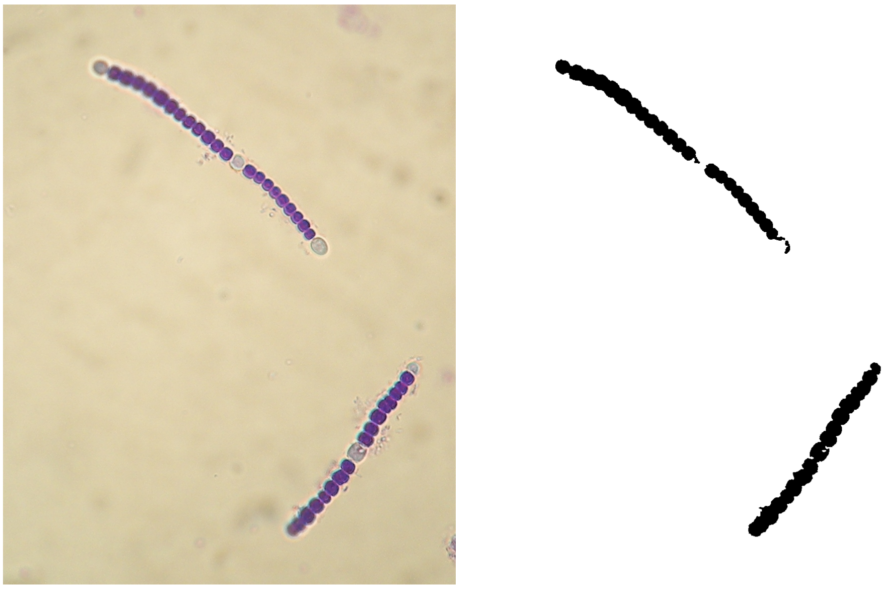


Fig. 1. (Left) Image with two filaments of heterocystous cyanobacterium labeled with 0.02% crystal violet (the arrows indicates the presence of heterocysts, specialized cells, not labeled by crystal violet in our experimental labeling conditions: 0.02% crystal violet concentration and 5 minutes of coloration). (Right) Binarization of the image.

the number of cells in a image, or more exactly, the number of spots in the image that probably correspond to a cell. Since this is a empirical problem, instead of a formal definition of the problem, we will base our description in a case study. We start by considering the image of Fig. 1. Such image correspond to filamentous cyanobacterium and has been taken by using the number of cells within each filament. It is very important to calculate the ratio of the number of heterocysts to non-differentiate cells, also called vegetative cell, as well as to calculate the distribution of filaments lengths (correlated to the total number of cells within each filament) within a population (composed of hundreds of filaments per each milliliter) of this cyanobacterium. In microbiological, studies for the determination of number of dividing cells related to the number of total cells are also very useful to calculate the growth rate in natural population of bacteria/cyanobacteria, heterocystous or not ([2, 7, 22, 25, 27, 29, 42]).

The target is to take the image as an input and provide a the number of cells as output and performing the different stages of the process by using Membrane Computing techniques. The stages are the following:

- **Stage 1: Binarization**. This fist stage consist on getting a new image from the original with only two colors (a binary image). In this stage we use a family

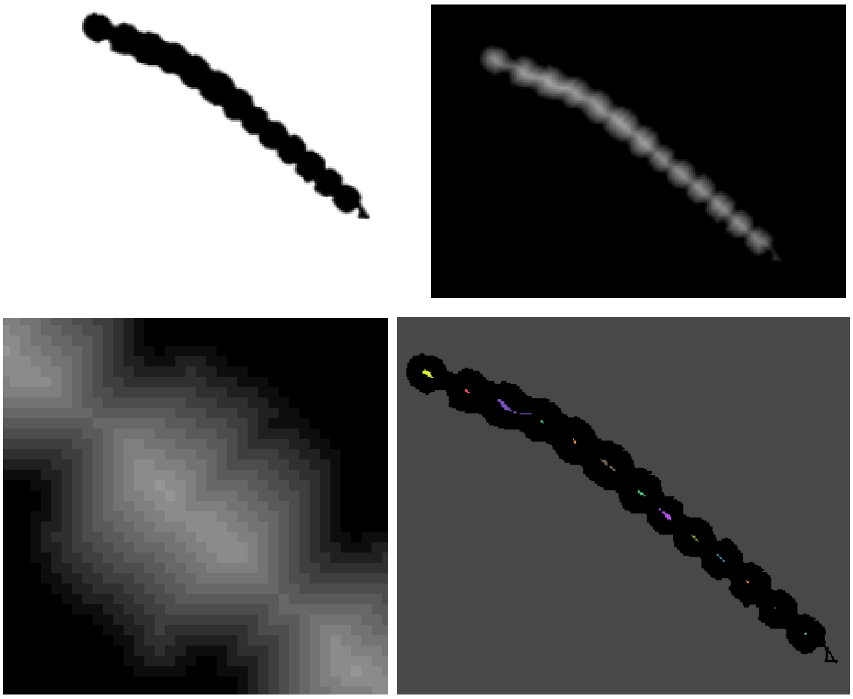


Fig. 2. The first image (top-left) is a detail of Figure 1 (left). The second and third ones (top-right and bottom-left) show details of the thinning process. Finally, the last image (bottom-right) shows a set of pixels inside each individual cells. Different cells are marked with different colors. The number of such different colors provides the number of cells.

of tissue-like P systems similar to the described one in [12, 32] for this aim. A result of this process can be seen in Figure 1 (right).

- **Stage 2: Segmentation**. This is the process that split the image in several *meaningful* regions. It is basic for the treatment of digital images and it is widely used in medical images for identifying the region of interest. We also perform this stage by using Membrane Computing techniques, namely, the algorithm described in [13].
- **Stage 3: Noise Reduction**. The images taken in a wet lab are usually far from having homogeneous regions. Due to the intrinsic nature of the biological research, in the image it is common tho find little spots that in a mechanic process of counting cells, can be easily taken as little cells when they are merely spots due to the noise. Obviously, removing such noise imply to take difficult decision and should be done carefully. We propose three different steps to eliminate noise from an image:

  - **Stage 3.1: Labeling black connected components**. The first step consists on determining connected components in the image. The set of pixels to be considered as a cell is a connected set of pixels. The connectivity problem among pixels and the study of connected components is a problem linking Digital Imagery and Algebraic Topology and it has been recently studied in the framework of Membrane Computing. For this step, we propose a partial application of the Membrane Computing algorithm proposed in [18].
  - **Stage 3.2: Calculating Areas**. This step starts after the identification of the connected components of the image. Our proposal is to use Membrane Computing techniques in order to determinate the *size* of the connected component. In our approach, the size of the spot is its number of pixels. This process is also performed by using the symport/antiport rules used in tissue like P systems and basically consists on counting the number of objects in the P system which represent the pixels of a concrete spot.
  - **Stage 3.3: Eliminating small components**. From the previous step, we have a number associated to each spot, representing its *size*. In this stage we will consider a *threshold* in order to decide if a spot is large enough to be considered *meaningful* or if the spot should be considered as noise, and hence, to be ignored. Obviously, the threshold depends on the experiment and it must be provided by the experts.
- **Stage 4: Counting Cells**. The three previous stages can be considered as preprocessing of the image. The algorithm for counting cells starts now. The key point in this stage is to consider the geometry that a cell usually shows in an image taken form a microscopy in a wet lab. Such images usually show the cells as convex spots and the image of multicellular beings usually shows a wavy border. An appropriate process of thinning, inspired in other thinning process[2] with Membrane Computing techniques, produces little isolated set of point for each cell (see Figure 2 (bottom-right)).
- **Stage 5: Output**. In the last stage, the isolated set of pixels that represent the cells are codified as an appropriate set of objects of the alphabet in the output membrane of the corresponding P system. Counting these sets of objects provides the number of cells in the image

3.1 Implementation

Inspired in the families of tissue-like P systems that perform the stages of the process of counting cells, a software tool has been implemented by using CUDA™, (Compute Unified Device Architecture) [28, 31]. CUDA™ is a general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processors Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU.

[2] See these proceedings

The experiments have been performed on a computer with a CPU AMD Athlon II x4 645, which allows to work with four cores of 64 bits to 3.1 GHz. The computer has four blocks of 512KB of L2 cache memory and 4 GB DDR3 to 1600 MHz of main memory. The used graphical card (GPU) is an NVIDIA Geforce GT240 composed by 12 *Stream Processors* with a total of 96 cores to 1340 MHz. It has 1 GB DDR3 main memory in a 128 bits bus to 700 MHz. So, the transfer rate obtained is by 54.4 Gbps. The used Constant Memory is 64 KB and the Shared Memory is 16 KB. Its Compute Capability level is 1.2 (from 1.0 to 2.1). The implementation deals with N blocks of threads for the complete image in our GPU of 96 cores.

3.2 Example

Figures 1 and 2 shows several details of an illustrative example of the process. The original image in Fig. 1 shows a filamentous cyanobacterium able to differentiate heterocysts (the arrows indicates the presence of heterocysts, not labeled by crystal violet in our experimental conditions) when grown on mineral medium (so called BG0) in the absence of combined nitrogen (biological identification at the level of genus is under work). In these conditions the biological specimen (the filamentous cyanobacterium) is able to utilize atmospheric nitrogen as a source of nitrogen to synthetizise its own biochemical components (amino acids, proteins etc); the first major steps in this utilization of atmospheric nitrogen occurs in heteocysts, differentiate cells within the filament [36]. The number of cells found in the image from Fig. 2 is 13.

4 Conclusions

The development of experimental sciences where a big amount of data is stored as digital images needs of powerful software which helps the researcher to understand the studied processes. In particular, Microbiology significantly benefits if automated image analysis is used by microbiologists in their professional activities; furthermore the interplay between microbiologists, mathematicians and engineers in this field could be helpful in developing new opportunities within *old* software, or ,even, to generate new software more appropriate for different microbiological task.

In this respect, Membrane Computing devices have features that make them suitable for the study of digital images, as the encapsulation of the information and its treatment in parallel. This paper reports how an appropriate combination of families of tissue-like P systems can solve the problem of counting cells. The study of how these or other families can be combined for solving more problems from Microbiology (or from other experimental sciences) is a challenge for the next years.

Acknowledgements

DDP and MAGN acknowledge the support of the project 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. IS and IA are grateful to Dr. Tech. Jyrki Selinummi (Department of Signal Processing, Tampere University of Technology, Finland) for very useful and kind discussion concerning digital image analysis.

References

1. Adeoye, O.S.: A survey of emerging biometric technologies. International Journal of Computer Applications 9(10), 1–5 (2010)
2. Agawin, N.S., Agusti, S.: Abundance, frequency of dividing cells and growth rates of synechococcus sp. (cyanobacteria) in the stratified northwest mediterranean sea. Journal of Plankton Research 19(11), 1599–1615 (1997)
3. Ayache, N.: Medical image analysis and simulation. In: Shyamasundar, R.K., Ueda, K. (eds.) ASIAN. Lecture Notes in Computer Science, vol. 1345, pp. 4–17. Springer (1997)
4. Belyaev, N., Paavilainen, S., Korpela, T.: Characterization of bacterial growth on solid medium with image analysis. Journal of Biochemical and Biophysical Methods 25(2-3), 125–32 (1992)
5. Bloem, J., Veninga, M., Shepherd, J.: Fully automatic determination of soil bacterium numbers, cell volumes, and frequencies of dividing cells by confocal laser scanning microscopy and image analysis. Applied and Environmental Microbiology 61(3), 926–936 (1995)
6. Carnero, J., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A.: Designing tissue-like P systems for image segmentation on parallel architectures. In: Martínez-del-Amor, M.A. *al.* (eds.) Ninth Brainstorming Week on Membrane Computing. pp. 43–62. Fénix Editora, Sevilla, Spain (2011)
7. Carpenter, E., Campbell, L.: Diel patterns of cell division and growth rates of synechococcus spp. in long island sound. Marine Ecology Progress Series 47, 179–183 (1988)
8. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Implementing P systems parallelism by means of GPUs. In: Păun, Gh. *et al.* (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5957, pp. 227–241. Springer, Berlin Heidelberg (2009)
9. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a P system based efficient solution to SAT by using GPUs. Journal of Logic and Algebraic Programming 79(6), 317–325 (2010)
10. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics 11(3), 313–322 (2010)
11. Chávez de Paz, L.E.: Image analysis software based on color segmentation for characterization of viability and physiological activity of biofilms. Applied and Environmental Microbiology 75(6), 1734–9 (2009)

12. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Thresholding of 2D images with cell-like P systems. Romanian Journal of Information Science and Technology 13(2), 131–140 (2010)
13. Christinal, H.A., Díaz-Pernil, D., Real, P.: Segmentation in 2D and 3D image using tissue-like P system. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Lecture Notes in Computer Science, vol. 5856, 169–176 (2009)
14. Christinal, H.A., Díaz-Pernil, D., Real, P.: Region-based segmentation of 2D and 3D images with tissue-like P systems. Pattern Recognition Letters 32(16), 2206 – 2212 (2011)
15. Collins, R., Lipton, A., Kanade, T.: Introduction to the special section on video surveillance. Pattern Analysis and Machine Intelligence, IEEE Transactions on 22(8), 745 –746 (aug 2000)
16. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P.: A bio-inspired software for segmenting digital images. In: Nagar, A.K., Thamburaj, R., Li, K., Tang, Z., Li, R. (eds.) Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications BIC-TA. vol. 2, pp. 1377 – 1381. IEEE Computer Society, Beijing, China (2010)
17. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P.: Designing a new software tool for digital imagery based on P systems. Natural Computing pp. 1–6 (2011)
18. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Real, P., Sánchez-Canales, V.: Computing homology groups in binary 2D imagery by tissue-like P systems. Romanian Journal of Information Science and Technology 13(2), 141–152 (2010)
19. Fero, M., Pogliano, K.: Automated quantitative live cell fluorescence microscopy. Cold Spring Harb Perspectives in Biology 2(8), (2010)
20. Gimel'farb, G., Nicolescu, R., Ragavan, S.: P systems in stereo matching. In: Real, P. *et al.* (eds.) Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol. 6855, pp. 285–292. Springer Berlin / Heidelberg (2011)
21. Gimel'farb, G.L.: Probabilistic regularisation and symmetry in binocular dynamic programming stereo. Pattern Recognition Letters 23(4), 431–442 (2002)
22. Hagström, A., Larsson, U., Hörstedt, P., Normark, S.: Frequency of dividing cells, a new approach to the determination of bacterial growth rates in aquatic environments. Applied and Environmental Microbiology 37(5), 805–12 (1979)
23. Heydorn, A., Nielsen, A.T., Hentzer, M., Sternberg, C., Givskov, M., Ersbøll, B.K., Molin, S.: Quantification of biofilm structures by the novel computer program COMSTAT. Microbiology 146(10), 2395–2407 (2000)
24. Ionescu, M., Sburlan, D.: On p systems with promoters/inhibitors. Journal of Universal Computer Science 10(5), 581–599 (2004)
25. Joung, S.H., Kim, C.J., Ahn, C.Y., Jang, K.Y., Boo, S.M., Oh, H.M.: Simple method for a cell count of the colonial cyanobacterium, microcystis sp. Journal of Microbiology 44(5), 562–5 (2006)
26. Lehmussola, A., Ruusuvuori, P., Selinummi, J., Huttunen, H., Yli-Harja, O.: Computational framework for simulating fluorescence microscope images with cell populations. IEEE Transactions on Medical Imaging pp. 1010–1016 (2007)
27. Li, W.K.W., Dickie, P.M.: Relationship between the number of dividing and non-dividing cells of cyanobacteria in north atlantic picoplankton. Journal of Phycology 27(5), 559–565 (1991)

28. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue 6, 40–53 (2008)
29. Nielsen, S.L.: Size-dependent growth rates in eukaryotic and prokaryotic algae exemplified by green algae and cyanobacteria: comparisons between unicells and colonial growth forms. Journal of Plankton Research 28(5), 489–498 (2006)
30. Ogawa, M., Tani, K., Yamaguchi, N., Nasu, M.: Development of multicolour digital image analysis system to enumerate actively respiring bacteria in natural river water. Journal of Applied Microbiology 95(1), 120–128 (2003)
31. Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C.: GPU Computing. Proceedings of the IEEE 96(5), 879–899 (2008)
32. Peña-Cantillana, F., Díaz-Pernil, D., Berciano, A., Gutiérrez-Naranjo, M.A.: A parallel implementation of the thresholding problem by using tissue-like P systems. In: Real, P. *et al.* (eds.) CAIP (2). Lecture Notes in Computer Science, vol. 6855, pp. 277–284. Springer (2011)
33. Peña-Cantillana, F., Díaz-Pernil, D., Christinal, H.A., Gutiérrez-Naranjo, M.A.: Implementation on CUDA of the smoothing problem with tissue-like P systems. International Journal of Natural Computing Research 2(3), 25–34 (2011)
34. Pernthaler, J., Alfreider, A., Posch, T., Andreatta, S., Psenner, R.: In situ classification and image cytometry of pelagic bacteria from a high mountain lake (Gossenköllesee, Austria). Applied and Environmental Microbiology 63(12), 4778–83 (1997)
35. Pettipher, G., Rodrigues, U.M.: Semi-automated counting of bacteria and somatic cells in milk using epifluorescence microscopy and television image analysis. Journal of Applied Microbiology 53(3), 323–329 (1982)
36. Sarchizian, I., Ardelean, I.I.: Axenic culture of a diazotrophic filamentous cyanobacterium isolated from mesothermal sulphurous springs (Obanul Mare - Mangalia). Romanian Journal of Biology - Plant Biology 55(1), 47–53 (2010)
37. Selinummi, J., Ruusuvuori, P., Podolsky, I., Ozinsky, A., Gold, E., Yli-Harja, O., Aderem, A., Shmulevich, I.: Bright field microscopy as an alternative to whole cell fluorescence in automated analysis of macrophage images. PLoS ONE 4(10), 9 (2009)
38. Selinummi, J., Seppälä, J., Yli-Harja, O., Puhakka, J.A.: Software for quantification of labeled bacteria from digital microscope images by automated image analysis. Biotechniques 39(6), 859–863 (2005)
39. Sheeba, F., Thamburaj, R., Nagar, A.K., Mammen, J.J.: Segmentation of peripheral blood smear images using tissue-like P systems. Bio-Inspired Computing: Theories and Applications (BIC-TA), 2011 Sixth International Conference on 0, 257–261 (2011)
40. Shopov, A., Williams, S.C., Verity, P.G.: Improvements in image analysis and fluorescence microscopy to discriminate and enumerate bacteria and viruses in aquatic samples. Aquatic Microbial Ecology 22(2), 103–110 (2000)
41. Van Wambeke, F.: Enumeration and size of planktonic bacteria determined by image analysis coupled with epifluorescence. Annales de l'Institut Pasteur Microbiology 139(2), 261–72 (1988)
42. Yamamoto, Y., Shiah, F.K.: Relationship between cell growth and frequency of dividing cells of microcystis aeruginosa. Plankton Benthos Res 5(4), 131–135 (2010)
43. Yang, X., Beyenal, H., Harkin, G., Lewandowski, Z.: Quantifying biofilm structure using image analysis. Journal of Microbiological Methods 39(2), 109–119 (2000)

General Topologies and P Systems

Erzsébet Csuhaj-Varjú[1], Marian Gheorghe[2,3], and Mike Stannett[2]

[1] Department of Algorithms and Their Applications
Faculty of Informatics, Eötvös Loránd University,
Pázmány Péter st. 1/c, Budapest, 1117, Hungary
csuhaj@inf.elte.hu

[2] Department of Computer Science, The University of Sheffield
Regent Court, 211 Portobello, Sheffield S1 4DP, United Kingdom
{M.Stannett, M.Gheorghe}@dcs.shef.ac.uk

[3] Department of Computer Science, University of Piteşti
Str Targu din Vale, Piteşti, Romania

Summary. In this paper we investigate the use of general topological spaces as control mechanisms for membrane systems. For simplicity, we illustrate our approach by showing how arbitrary topologies can be used to study the behaviour of membrane systems with rewrite and communication rules.

1 Introduction

Membrane computing has emerged in the last more than ten years as a vigorous research field as part of natural computing or unconventional computing. It is a nature-inspired computational paradigm including a large variety of models, called *membrane systems*, well-investigated from a computational perspective, especially with respect to their computational power and complexity aspects [9]. A number of promising applications, mainly in biology, but also in distributed computing, linguistics and graphics [1], have been identified and described.

The key features of a membrane system are a set of *compartments* (called *regions*) delimited by *membranes*, *multisets of objects* contained in these regions, *transformation and communication rules* describing interactions between objects, and a *strategy* for evolving the system. This basic model is inspired by standard models of the structure and functions of a typical eukaryotic cell, comprising multiple compartments containing localised biochemical materials and reactions: various chemical entities with different levels of complexity react under specified circumstances to produce new biochemicals supporting the cell's life and metabolism, and these may or may not be transported to other compartments depending on context. Many variants of membrane system have been considered, some using different types of biochemical agent and interaction, others using various types of structural organisation for the compartments and their connections [9].

Membrane systems introduce in a very natural way a specific topology on the system described, in which membranes delimit compartments containing local objects and interaction rules, together with specific links between compartments. These links describe communication channels allowing adjacent compartments to exchange chemicals. Although this topology is flexible enough to cope with the challenge of modelling various natural or engineering systems, there are cases when a finer grain topological structure is required. In a series of papers, J.-L. Giavitto and his collaborators have investigated the use of topological transformations applied to various data structures, where algebraic topology helps in defining the appropriate data sets selected to be transformed [2]. The use of this approach to model various elements and transformations occurring in membrane computing has been investigated in [4], while concepts related to a spatial computing programming paradigm, which permit the definition and handling of a sort of geometry, have been described in the context of the unconventional programming language, MGS [7].

In this paper we investigate the use of topological spaces as *control mechanisms* for membrane systems. While the algebraic topological approach shows how the membrane structure and its basic operations with multisets can be represented, here we use a topological space as a framework to control the evolution of the system with respect to a family of open sets that is associated with each compartment. This approach produces a fine grain description of local operations occurring in each compartment by restricting the interactions between objects to those from a given neighbourhood. This initial study shows the influence of an arbitrary topology on the way basic membrane systems compute. In future work (cf. Sect. 5) we aim to investigate the role of more specific topologies, their impact on other types of membrane system, and their applications in solving/approaching various problems.

2 Basic notations and definitions

We briefly recall basic notions concerning P systems. For more details on these systems and on P systems in general, we refer to [8, 9]. A basic evolution-communication P system (P system for short) of degree n is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_0)$$

where

1. O is a finite alphabet of symbols called objects;
2. μ is a membrane structure consisting of n membranes that are labelled (in a one-one manner) with elements from a given alphabet A; these membranes are organised in a hierarchical way, like a tree, with the top membrane (root) called the *skin membrane*, and the bottom ones (leaves) called *elementary membranes*;

3. for each $1 \leq i \leq n$, $w_i \in O^*$ is a multiset of objects associated with the region i (this is the region delimited by membrane i, but not including the subregions delimited by i's children);
4. for each $1 \leq i \leq n$, R_i is a finite set of rules associated with the region i, of the form $u \to (v_1, tar_1) \ldots (v_m, tar_m)$, where $u \in O^+$, $v_j \in O$ and $tar_j \in \{in, out, here\}$ $(1 \leq j \leq m)$; when tar_j is *here*, we write simply v_j in place of (v_j, tar_j);
5. i_0 is the label of an elementary membrane of μ that identifies the corresponding output region.

A P system is interpreted dynamically as a computational device comprising a set of n hierarchically nested membranes that identify n distinct regions (the membrane structure μ), where each region $i = 1, \ldots, n$ contains a multiset of *objects* (w_i) and a finite set of *evolution rules* (R_i) of the form $u \to (v_1, tar_1) \ldots (v_m, tar_m)$. This rule removes multiset u from region i, and then adds each multiset v_j $(1 \leq j \leq m)$ to the multiset of objects in the corresponding target region tar_j.

- If tar_j does not appear in the notation (by convention this occurs when the target is *here*), then v_j remains in membrane i;
- If tar_j is *out*, then v_j is sent to the parent membrane of i; if i is the skin membrane then v_j is sent out of the system;
- If tar_j is *in*, then v_j is sent to one of the inner membranes of i (if there is more than one child, the target is chosen non-deterministically);
- The *in* target can be replaced by a precisely defined destination region. If region k is a child of i and tar_j is k, then v_j is sent to k.

A computation of the system is obtained by applying the available rewrite rules in a non-deterministic maximally parallel manner[4], where each region i initially contains the corresponding finite multiset w_i.

A computation is considered successful when it starts from the initial configuration and reaches a configuration where no further rules can be applied. Its result, a natural number, is obtained by counting the objects present in region i_0 on completion (other ways of interpreting the result of a P system computation are also considered in the literature [9]). Given the non-deterministic nature of P system computation, different runs of a given system may generate different results. For a given P system Π the set of numbers that can be computed is denoted $N(\Pi)$.

Recall that rewrite rules are of the form $u \to (v_1, tar_1) \ldots (v_m, tar_m)$, where u is a multiset. If in each of the rules in Π the multiset u contains only a single object, then Π is called a *P system with non-cooperative rules*; otherwise it is a *P system with cooperative rules.* When $tar_j = in$ the rule is said to have *arbitrary target*, and when $tar_j = in_k$ for a specific region k, it has a *selected* target.

[4] A simultaneous application of rewrite rules is *non-deterministic maximally parallel* provided the applied rules are chosen non-deterministically (possibly with repetition), and there are insufficient resources to trigger the simultaneous application of any additional rule.

2.1 Topological conventions

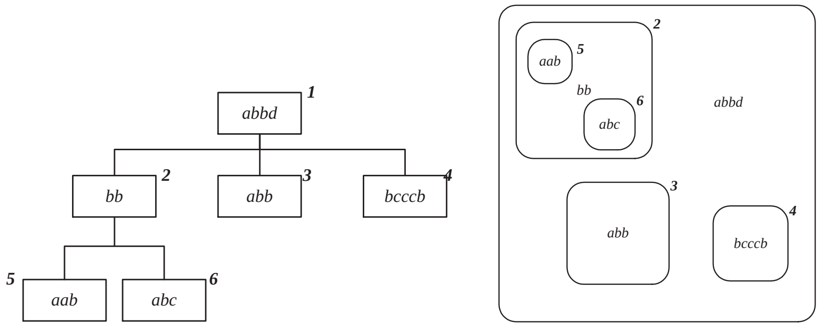
Our notation will generally follow that of [10]. Given any non-empty set X, its power set will be denoted $\wp X$. We write $\varnothing$ for the empty set. A *topology* on X is any subset of $\wp X$ containing both $\varnothing$ and X, which is closed under arbitrary unions and finite intersections; the members of $\mathcal{T}$ are *open* (or $\mathcal{T}$*-open* where ambiguity might otherwise arise). The topology $\{\varnothing, X\}$ is the *indiscrete* topology on X; the topology in which every singleton $\{x\} \in \wp X$ is open is the *discrete* topology. An *open cover* of $A \subseteq X$ is a subset of $\mathcal{T}$ whose union contains A.

The complement of an open set is *closed*. The *closure* $\overline{A} = \mathrm{Cls}_X(A)$ of a set $A \subseteq X$ is the intersection of all closed sets containing A; it is the smallest such set. The *interior* $A^\circ = \mathrm{Int}_X(A)$ of A is the union of all open sets contained in A; it is the largest such set. The difference between the closure and interior of a set is its *boundary*, $\partial A = \overline{A} \setminus A^\circ$.

Any topology $\mathcal{T}$ can also be regarded as a partially ordered set (*poset*) ordered by set inclusion. If $(Y, \leq)$ is a poset, an *order embedding* of Y in $\mathcal{T}$ is an injection $\imath \colon Y \to \mathcal{T}$ such that $y_1 \leq y_2$ if and only if $\imath(y_1) \leq \imath(y_2)$.

3 Control structures

For the purposes of this paper, a P system can be regarded structurally as a tree whose nodes are the *membranes*, together with a function mapping each node p in Π to a corresponding multiset over A. This multiset tells us how many copies of each object lie in the region situated between the membrane and its internal sub-membranes; see Fig. 1.

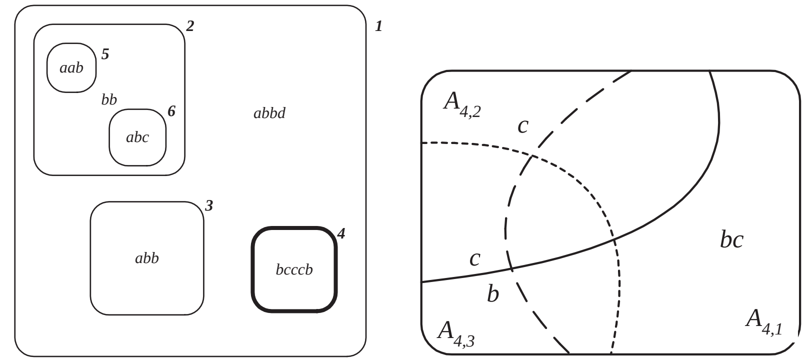


(a) tree (b) nested membranes

Fig. 1. A generic P system structure represented as (a) a tree; (b) a set of nested membranes.

In each membrane and in any computation step it is assumed that all the objects present in the corresponding region can freely interact according to the set of rules available in that region. Maximal parallelism also implies that all the objects that might take part in various interactions must interact (each object takes part in at most one interaction). While this scheme is easy to implement, it distorts to some extent the biological intuition that interactions are *local*. It is not enough that two chemicals are present in a cell, they must also be located close to one another, but the regions of a P system are not inherently associated with any notion of separation distance. We will therefore order-embed the membranes of the P system as open sets within an essentially arbitrary topology, and use (finite) open covers to provide an indication of the distance between two objects. We then consider how the choice of topology affects the computations that can be implemented.

In general the members of an open cover need not be disjoint. Region 4 of Fig. 1 contains the multiset *bcccb*. Figure 2 illustrates a covering of this region by three open sets: $A_{4,1}$, $A_{4,2}$ and $A_{4,3}$. The open set $A_{4,2}$ contains *cc*, and each of the others contains *bc*. Initially we only consider open covers of *regions*; subregions of the enclosing membrane will be equipped with covers in their own right.



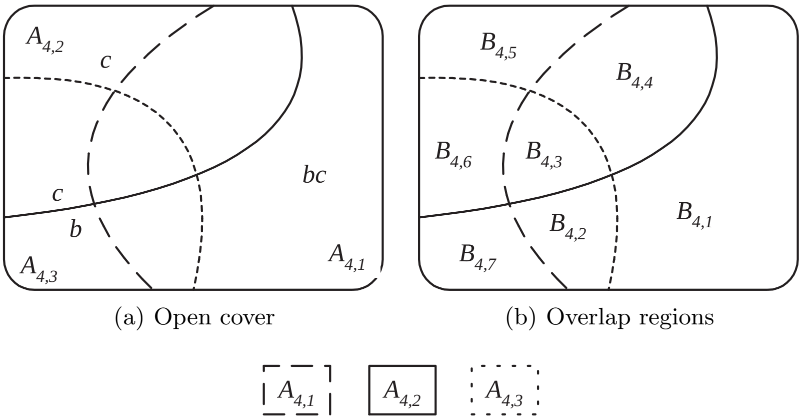
(a) Nested membranes (region 4 highlighted)

(b) Open covering of region 4

Fig. 2. Covering of region 4 by open sets.

The *topologically controlled computation* that takes place with respect to these open sets is defined as follows: *rules associated with membrane i are enabled if and only if there is a member of the open cover which contains all of the participating objects*. If any target of an enabled rule is *here*, the associated products should then be placed back into the same open set (if the initial objects lie in more than one member of the cover, we choose one at random and place the associated results there; they need not be injected back into the intersection). Otherwise if the target is *tar* (where *tar* is assumed to carry its own open cover), the output will be placed in an arbitrary member of *tar*'s cover.

Despite the intrinsically local nature of controlled computation, the locus of computation can migrate from one compartment to another one via non-empty overlap regions, as the following example illustrates. Figure 3 shows the disjoint parts, $B_{4,1} - B_{4,7}$, of region 4's cover. These are all of the form $U \backslash V$ where U and V are open; for example $B_{4,3} = (A_{4,1} \cap A_{4,2} \cap A_{4,3}) \backslash \varnothing$, and $B_{4,1} = A_{4,1} \backslash (A_{4,2} \cup A_{4,3})$.



(a) Open cover

(b) Overlap regions

(c) Key to boundaries

Fig. 3. The finite covering of region 4 and its disjoint overlap regions.

Suppose, then, that region 4 has the following rules associated with it:

$$r_1 : bc \to b; \quad r_2 : bcc \to c; \quad r_3 : cc \to c.$$

If we consider the system as a P system with no topological control in place, the following computations can take place:

1. $bc\,c\,cb \xRightarrow{r_1, r_1} bcb \xRightarrow{r_1} bb$
2. $bc\,cc\,b \xRightarrow{r_1, r_3} bcb \xRightarrow{r_1} bb$
3. $bc\,ccb \xRightarrow{r_1, r_2} bc \xRightarrow{r_1} b$

But when the open sets are in place computation path 3 is blocked, because none of the open sets ever contains bcc, whence r_2 cannot be triggered. We have the following two cases instead:

1' r_1 is applied in both $A_{4,1}$ and $A_{4,3}$ resulting in a copy of b in each of these open sets; if $b \in A_{4,3}$ is not in $A_{4,2} \cap A_{4,3}$ the computation stops here with bbc scattered across different open sets. If, on the other hand, $b \in A_{4,2} \cap A_{4,3}$ then the computation can continue; after applying r_1 in $A_{4,2}$ a copy of b is obtained in each of $A_{4,1}$ and $A_{4,2}$. In this case the result is the same as that obtained in (1);

2'. r_3, r_1 result in copies of $b \in A_{4,1}$ and $c \in A_{4,2}$; as in (1') this c can reside either in the intersection or outside it; in the first case r_1 can be applied again and b is computed (so that the result from (2) is obtained). Otherwise bbc will remain in the membrane unchanged.

Consider in particular the second case of the first step of (1'). After r_1 is applied in $A_{4,3}$ the result b can be considered to lie in $A_{4,2}$, whence (as suggested above) the locus of computation can migrate from $A_{4,3}$ to $A_{4,2}$ via their intersection. A similar situation occurs in (2') as well.

More generally, suppose that region i has a rule whose target is region j. We will allow the rule to be triggered only when the two regions are sufficiently close to one another (their boundaries must intersect: $\partial i \cap \partial j \neq \varnothing$). In this case, and provided all of the required components are available within a single member of i's cover, the rule can fire with the resulting multiset v_j being injected into an arbitrarily selected member of j's cover. In future work we plan to investigate what happens when this restriction is weakened, so that interactions can occur between non-neighbouring regions.

Definition 1 (Output of a controlled computation). *For a P system Π and associated topology $\mathcal{T}$ the set of numbers computed by Π when controlled by $\mathcal{T}$ will be denoted $N_{\mathcal{T}}(\Pi)$.* □

Having now defined controlled computation, we will address the following problems. In Sect. 4 we discuss the role of a control mechanism based on an associated topology and show how a general topology influences the computation for a basic class of P systems. In Sect. 5 we summarise our findings and discuss future research topics related to various topologies associated with classes of P systems.

4 Basic Results

We will first consider P systems with non-cooperative rules.

Lemma 1. *For any P system with non-cooperative rules and either arbitrary targets or selected targets, Π, and any associated topology $\mathcal{T}$, $N(\Pi) = N_{\mathcal{T}}(\Pi)$.*

Proof. In a P system with non-cooperative rules the left hand side of any rule has only one single object, hence no interactions are involved. In this case it is obvious the the topology $\mathcal{T}$ does not influence the computation for either P systems with arbitrary targets or selected targets, hence the result stated holds. □

For P systems with cooperative rules the situation is totally different and the topologies associated with them may lead to different computations and distinct results.

Lemma 2. *There is a P system with cooperative rules and either arbitrary or selected targets, Π, such that for any associated topology $\mathcal{T}$ where for at least one region not all the objects belong to the same open set, it follows that $N(\Pi) \neq N_{\mathcal{T}}(\Pi)$.*

Proof. Let us consider $\Pi = (O, \mu, w_1, w_2, R_1, R_2, i_0)$, where $O = \{a, b, c\}$, $\mu = [[]_2]_1$, $w_1 = ab$, $w_2 = \lambda$, $R_1 = \{ab \rightarrow c, c \rightarrow (c, in)\}$, $R_2 = \varnothing$, $i_0 = 2$. This system uses an arbitrary target, which, in this case, is the same as selected target, in_2. This P system computes c in two steps in the output region, 2. Any topology, $\mathcal{T}$, associated with Π that provides a cover for region 1 with more than an open set, must have an open set for a and another one for b and their intersection does not contain any of these two objects; otherwise, a and b will stay in the same open set. In this case the rule $ab \rightarrow c$ can not be applied and consequently c is never obtained in the output region, hence $N(\Pi) \neq N_{\mathcal{T}}(\Pi)$. □

Theorem 1. *For any P system with either arbitrary or selected targets, the computation and the topologically controlled computation are the same when non-cooperative rules are used and are not in general the same for cooperative rules.*

Proof. The proof is an immediate consequence of Lemmas 1 and 2. □

There are P systems with cooperative rules where the content of the regions can be matched against the open sets in such a way that the computation is equivalent to the computation of the original system. Indeed let us consider the problem of checking that a positive integer m is divided by another positive integer k. We propose a P system below which is an adaptation of the P system presented in [9].

Example 1. Let us consider $\Pi = (O, \mu, w_1, w_2, R_1, R_2, i_0)$, where $O = \{a, b, c, y, n\}$, $\mu = [[]_2]_1$, $w_1 = a^m b^k$, $w_2 = y$, $R_1 = \{r_1 : ab \rightarrow c, r_2 : ac \rightarrow b, r_3 : bc \rightarrow (n, in)\}$, $R_2 = \{yn \rightarrow n\}$, $i_0 = 2$.

In the first step at most k objects ab are replaced by the same number of objects c (using r_1 at most k times) and then objects ac are replaced by objects b (using r_2). If k divides m then the process will stop after h steps, where $m = kh$, and in membrane 2 will remain y; otherwise in membrane 1 the process of alternatively applying rules r_1 and r_2 will stop with some objects b and objects c and the rule r_3 can be used. In this case n is sent into region 2 and finally n is obtained in this region.

Now, if we aim to obtain the same results in region 2, i.e., y, when k divides m, or n otherwise, then we have to build the topology, $\mathcal{T}$, associated with Π in a certain way which is subsequently described. Region 2 is covered by only one single open set and region 1 will have an arbitrary number of open sets, $q > 1$, associated with. *Any two such open sets are disjoint.* The objects will be distributed as follows: the k b's will be randomly distributed in $q - 1$ of the q open sets, $b^{k_1}, \ldots, b^{k_{q-1}}$, $k_i \geq 0$ and $k_1 + \cdots + k_{q-1} = k$. If $m = kh + r$, then in each of the $q - 1$ open sets containing k_i b's, the number of a's is hk_i a's. If $r > 0$ then one more a will be consider in one of the $q - 1$ open sets with b's and the rest will be associated with the q^{th} open set. Clearly, in each of the $q - 1$ open sets the computation will go for h steps. In $q - 2$ of them it will be obtained either only b's or only c's; the open set with an additional a in it will end up after one more step with a mixture of b's and c's and the rule r_3 will push an n into membrane 2 and finally will get n in this membrane. Objects a's occurring in the q^{th} open set will remain there forever. It follows that $N(\Pi) = N_{\mathcal{T}}(\Pi)$. □

The question of whether the control structure introduced by a topology can be ignored, perhaps by using a more complex P system, is answered by the following result. This takes into account the interpretation of the outcome of the computation as being the number of objects, given by the size of the multiset, present in the output region.

Theorem 2. *For any P system, Π, and any associated topology, $\mathcal{T}$, there is a P system, Π', of the same degree with Π, such that $N_{\mathcal{T}}(\Pi) = N(\Pi')$.*

Proof. The idea of the proof is to construct a new P system such that objects belonging to a region adequately refer to objects of the open sets in the corresponding regions of the initial P system.

Let Π be a P system of degree n, $\Pi = (O, \mu, w_1, \dots, w_n, R_1, \dots, R_n, i_0)$, and $\mathcal{T}$ a topology associated with it. In order to build a new P system, Π', of degree n, a few preliminary notations are made. First, please observe that for each region i, $1 \le i \le n$, there exists a family of open sets $A_{i,1}, \dots, A_{i,k_i}$ covering it. In general these open sets are not disjoint and we describe the finest disjoint parts of the cover by considering either some intersections of open sets or the complement of an open set with respect to the rest of the open sets; it follows that there exists a finite set, denoted B_i, containing the sets $B_{i,1}, \dots, B_{i,m_i}$, such that $B_{i,j}$ denotes either $A_{i,l_1} \cap \dots \cap A_{i,l_j}$, $1 \le l_j \le k_i$ or $A_{i,j} \setminus (A_{i,1} \cup \dots \cup A_{i,j-1} \cup A_{i,j+1} \cup \dots \cup A_{i,k_i})$. The set of indexes of the above sets $B_{i,j}$ is denoted by C_i, i.e., $C_i = \{(i,j) \mid B_{i,j} \in B_i\}$. Each object, $a \in O$, of the multiset from region i belongs to a certain $B_{i,j}$. For each a from $B_{i,j}$, the following objects are considered, $a^{\alpha}, \alpha \in C_i$.

The P system Π', of degree n, is built as follows:

$$\Pi' = (O', \mu, w'_1, \dots, w'_n, R'_1, \dots, R'_n i_0) \ ,$$

where:

1. $O' = \{a^{\alpha} | a \in O, \alpha \in C_i, 1 \le i \le n\}$;
2. μ is the membrane structure of Π;
3. $w'_i = a_{i,1}^{(i,r_1)} \dots a_{i,p_i}^{(i,r_{p_i})}$, where $a_{i,j} \in B_{i,r_j}$, $1 \le j \le p_i$, for $w_i = a_{i,1} \dots a_{i,p_i}$, initial multiset of Π;
4. for each rule $a_{i,1} \dots a_{i,q_i} \to b_{i,1} \dots b_{i,s_i} \in R_i$, R'_i contains $a_{i,1}^{(i,r_1)} \dots a_{i,q_i}^{(i,r_{q_i})} \to b_{i,1}^{(i,s_1)} \dots b_{i,p_i}^{(i,s_{p_i})}$, $(i, r_j) \in C_i$, $1 \le j \le q_i$, $(i, s_j) \in C_i$, $1 \le j \le p_i$ when a target, t, appears on the right hand side of the rule from R_i, associated with an object $b_{i,j}$, then the target will point to any of the open sets $A_{t,j}$ of the target region t;
5. Π and Π' have the same output membrane, i_0.

The codification provided by Π' allocates, in a unique way, in every region, i, each object, a, to a specific open set, by "stamping" it with the corresponding index, $(i,j) \in C_i$, of the set $B_{i,j}$. Whenever a rule is applied, the resulted multiset is also composed of objects uniquely associated with certain open sets, either from the current region or from the target ones.

More precisely, when in region i of Π the current multiset is

$$u = a_1 \dots a_{q_1} a_{q_1+1} \dots a_{q_2} \dots a_{q_{e-1}} a_{q_{e-1}+1} \dots a_{q_e} z$$

and there are rules $\rho_1, \dots, \rho_e \in R'_i$, where $\rho_j : a_{q_{j-1}+1} \dots a_{q_j} \to b_{p_{j-1}+1} \dots b_{p_j}$, $q_0 = 0$, then $\rho_1, \dots, \rho_e$ are applied in a computation step, according to maximal parallelism semantics, to u with respect to topology $\mathcal{T}$.

If $u \Rightarrow_{\rho_1,\dots\rho_e} v$, with $v = b_1 \dots b_{p_1} \dots b_{p_{e-1}} \dots b_{p_e} z$ then each a_h, $q_{j-1}+1 \leq h \leq q_j$, belongs to a certain B_{i,r_h} included in the same open set $A_{i,j}$ where ρ_j is applied. Each of the objects b_h, $p_{j-1}+1 \leq h \leq p_j$, belongs to some B_{i,s_h} included in the same $A_{i,j}$ set.

In the P system Π', in region i, there is

$$u' = a_1^{(i,r_1)} \dots a_{q_1}^{(i,r_{q_1})} a_{q_1+1}^{(i,r_{q_1+1})} \dots a_{q_2}^{(i,r_{q_2})} \dots a_{q_{e-1}}^{(i,r_{q_{e-1}})} a_{q_{e-1}+1}^{(i,r_{q_{e-1}+1})} \dots a_{q_e}^{(i,r_{q_e})} z',$$

where $(i,j) \in C_i$, $j \in \{r_1, \dots, r_{q_e}\}$. The multiset z' consists of objects $(a^\alpha)^c$ for a^c occurring in z and $\alpha \in C_i$. There are rules $\rho'_1, \dots, \rho'_e$, where $\rho'_j : a_{q_{j-1}+1}^{(i,r_{q_{j-1}+1})} \dots a_{q_j}^{(i,r_{q_j})} \to b_{p_{j-1}+1}^{(i,s_{p_{j-1}+1})} \dots b_{p_j}^{(i,s_{p_j})}$, which are applied in a maximal parallel manner to u'. If $u' \Rightarrow_{\rho'_1,\dots\rho'_e} v'$, then

$$v' = b_1^{(i,s_1)} \dots b_{p_1}^{(i,s_{p_1})} \dots b_{p_{e-1}}^{(i,s_{p_{e-1}})} \dots b_{p_e}^{(i,s_{p_e})} z',$$

where $(i,j) \in C_i$, $j \in \{s_1, \dots, s_{p_e}\}$.

The above construction proves that $u \Rightarrow_{\rho_1,\dots\rho_e} v$ in Π if and only if $u' \Rightarrow_{\rho'_1,\dots\rho'_e} v'$ in Π'. This shows that the same number of symbols are engaged in any computation step in Π and Π', hence these P systems compute the same number of symbols in i_0. □

From the above proof it is clear that the numbers of objects and rules used by the P system Π' are both significant compared to those of Π. The next result provides lower and upper bound limits for these two parameters. We need a few more notations to describe the result.

For a finite set X, let us denote by $card(X)$, the number of elements of X. With respect to the proof of Theorem 2, the following notations are introduced: K is the number of elements of the set O, n is the degree of the two P systems, Π and Π'; given that for each region i, $1 \leq i \leq n$, the number of sets $B_{i,j}$ is m_i, let us denote $m = min\{m_i \mid 1 \leq i \leq n\}$, $M = max\{m_i \mid 1 \leq i \leq n\}$, $p = min\{|x|, |y| \mid \text{all } x \to y \in R_i, 1 \leq i \leq n\}$ and $P = max\{|x|, |y| \mid \text{all } x \to y \in R_i, 1 \leq i \leq n\}$; if g_i is the maximum number of neighbours that appear in the rules of R_i, then $g = min\{g_i \mid 1 \leq i \leq n\}$; finally we have $Q = card(R_1 \cup \dots \cup R_n)$. With these notations we can formulate the following result.

Corollary 1. *For any P system Π and any associated topology $\mathcal{T}$, define Π' and the associated notation as above. Then*

(i) $Kmn \leq card(O') \leq KMn$;

(ii)$Qm^p(min\{m,g\})^p \leq card(R'_1 \cup \cdots \cup R'_n) \leq QM^P(M+n-1)^P$.

Proof. Part (i) follows from the fact that for each object $a \in O$, distinct instances are created for each of the n membranes and in every region i $(1 \leq i \leq n)$, and each set $B_{i,j}$ $(1 \leq j \leq m_i)$. Hence, $card(O')$ is bounded between Kmn and KMn.

To prove (ii), we observe that for each rule $x \to y \in R_i$, the following rules are added to R'_i, $x^\alpha \to y^\beta$, $\alpha \in C_i$, $\beta \in C_i \cup C_{j_1} \cup \cdots \cup C_{j_i}$, where $j_1, \ldots, j_i$ are neighbours of i where objects of y^β can go to. The left hand side, x^α, will have elements from any of the m_i sets, $B_{i,j}$, hence the lower and upper bounds are m^p and M^P, respectively. Each of the right hand side elements of y^β should belong to either one of the $B_{i,j}$ sets or to one of the neighbours of i, maximum $n-1$, so the lower and upper bounds are $(min\{m,g\})^p$ and $(M+n-1)^P$, respectively. We can then get the two boundaries of $card(R'_1 \cup \cdots \cup R'_n)$. □

5 Summary and Conclusions

In this paper we have investigated the use of general topological spaces to control local interactions in basic membrane systems. This approach produces a fine grain description of local operations occurring in each compartment by restricting the interactions between objects to those from a certain vicinity. In our future work we aim to investigate the role of more specific topologies, their impact on other types of membrane systems and their applications to various problems. In particular:

1. By construction, P systems have a tree-like nested membrane structure. Given the topological embeddings used in this paper, it is no longer clear whether this structure is relevant; the same proofs appear to work for different underlying graph structures with some adjustments.
2. It would be interesting to study the robustness of P systems with respect to different topologies. How much we can change the topology while still obtaining the same or almost the same computed set of numbers? To what extent can locality be refined starting from a given topology and changing it?
3. If we restrict attention to classes of control space (Tychonov spaces, compact Hausdorff spaces, metric spaces, etc) for which a wide range of topological results are available, can these results be applied to produce associated characterisations of controlled computability?

Acknowledgements. MPS and MG were partially supported under the Royal Society International Exchanges Scheme (ref. IE110369); MG was also partially supported by the project MuVet, Romanian National Authority for Scientific Research (CNCS – UEFISCDI) grant number PN-II-ID-PCE-2011-3-0688.

References

1. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing.* Springer, Berlin, 2006.

2. J.-L. Giavitto: Topological collections, transformations and their applications to the modeling and simulation of dynamical systems. *The 14-th International Conference on Rewriting Techniques and Applications*, LNCS 2706, Springer, 2003, 208–233.
3. J.-L. Giavitto, O. Michel: MGS: A rule-based programming language for complex objects and collections. *Electronic Notes in Theoretical Computer Science*, 59 (2001), 286–304.
4. J.-L. Giavitto, O. Michel: The topological structures of membrane computing. *Fundamenta Informaticae*, 49 (2002), 123–145.
5. J.-L. Giavitto, H. Klaudel, F. Pommereau: Qualitative modelling and analysis of regulations in multi-cellular systems using Petri nets and topological collections. Proceedings Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC) 2010 (G. Ciobanu, M. Koutny eds.), EPTCS, 40 (2010).
6. J.-L. Giavitto, H. Klaudel, F. Pommereau: The topological ttructures of membrane computing. *Theoretical Computer Science*, submitted.
7. J.-L. Giavitto, A. Spicher: Topological rewriting and the geometrization of programming. *Physica D*, 237 (2008), 1302–1314.
8. Gh. Păun: Computing with membranes. *Journal of Computer and System Science*, 61 (2000), 108–143.
9. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing.* Oxford Univ. Press, 2010.
10. S. Willard: *General Topology.* Dover Publications Inc., Mineola, NY, 2004.

Skeletonizing Images by Using Spiking Neural P Systems

Daniel Díaz-Pernil[1], Francisco Peña-Cantillana[2],
Miguel A. Gutiérrez-Naranjo[2]

[1]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics
University of Sevilla
sbdani@us.es

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

Summary. Skeletonizing an image is representing a shape with a small amount of information by converting the initial image into a more compact representation and keeping the meaning features. In this paper we use spiking neural P systems to solve this problem. Based on such devices, a parallel software has been implemented on the GPU architecture. Some real-world applications and open lines for future research are also presented.

1 Introduction

Computer vision [32] is probably one of the challenges for computer scientists in the next years. This flourishing research area needs contributions from many other scientific areas as artificial intelligence, pattern recognition, signal processing, neurobiology, psychology or image processing among others. It concerns with the automated processing of images from the real world to extract and interpret information on a real time basis. From a computational point of view, a digital image is a function from a two dimensional surface which maps each point in the surface to a set of features as bright or color. The different treatments of such mappings (digital images) provide a big amount of current applications in computer vision as optical character recognition (OCR), biometrics, automotive safety, surveillance or medical imaging.

In this paper we focus on the problem of skeletonizing an image. Skeletonization is one of the approaches for representing a shape with a small amount of information by converting the initial image into a more compact representation and keeping the meaning features. The conversion should remove redundant information, but it should also keep the basic structure. Skeletonization is usually considered as a pre-process in pattern recognition algorithms, but its study is also interesting by itself for the analysis of line-based images as texts, line drawings, human fingerprints or cartography.

Many problems in the processing of digital images have features which make it suitable for techniques inspired by nature. One of them is that the treatment of the image can be parallelized and locally solved. Regardless how large is the picture, the process can be performed in parallel in different local areas of it. Another interesting feature is that the local information needed for a pixel transformation can also be easily encoded in the data structures used in Natural Computing. In the literature, we can find many examples of the use of Natural Computing techniques for dealing with problems associated to the treatment of digital images. One of the classic examples is the use of cellular automata [28, 31]. Other efforts are related to artificial neural networks as in [9, 35]. In this paper, we use *spiking neural P systems*.

Spiking neural P systems (SN P systems, for short) were introduced in [16] as a new class of distributed and parallel computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons. SN P systems are the third model of computation in the framework of Membrane Computing[1], together with the cell-like model [25] inspired by the compartmental structure and functioning of a living cell and the tissue-like model [17], based on intercellular communication and cooperation between cells in a tissue.

Recently, Membrane Computing techniques have been used for solving problems from Digital Image. Different P systems models have been used for dealing with images, as in [3] where cell-like P systems are used for computing the thresholding of 2D images; [4, 5, 23, 24] where tissue-like P systems are used, or even [10], where the *symmetric dynamic programming stereo* (SDPS) algorithm [11] for stereo matching was implemented by using simple P modules with duplex channels. To the best of our knowledge, this is the first time in which SN P systems are used for dealing with images.

In a similar way that other applications of P systems, the theoretical advantages of the Membrane Computing techniques for computer vision need a powerful software and hardware for an effective implementation. In this paper, we also present a parallel software developed by using a device architecture called CUDA™, (Compute Unified Device Architecture). CUDA™ is a general purpose parallel computing architecture that allows the parallel NVIDIA[2] Graphics Processors Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU. GPUs constitute nowadays a solid alternative for high performance computing, and the advent of CUDA allows programmers a friendly model to accelerate a broad range of applications. The way GPUs ex-

[1] We refer to [26] for basic information in this area, to [27] for a comprehensive presentation and the P system web page `http://ppage.psystems.eu`, for the up-to-date information.

[2] `http://www.nvidia.com`.

ploit parallelism differs from multi-core CPUs, which raises new challenges to take advantage of its tremendous computing power. GPU is especially well-suited to address problems that can be expressed as data-parallel computations.

The paper is organized as follows: Firstly, we present the restricted model of SN P systems used in the paper and recall the Guo & Hall algorithm for skeletonizing images. In Section 4, the design of the SN P system for skeletonizing images is presented. Next, we show illustrative examples of the use of our implementation. Finally, Section 6 is dedicated to conclusions and future work.

2 Spiking Neural P Systems

SN P systems can be viewed as an evolution in Membrane Computing corresponding to a shift from *cell-like* to *neural-like* architectures. In SN P systems the processing elements are called *neurons* and are placed in the nodes of a directed graph, called the *synapse graph.* The computation is performed by sending electrical impulses among the neurons through the synapses. Such electrical impulses are encoded via a single object type, namely the *spike*, which is placed in the neurons. The number of copies of such object determines the electrical charge of the neuron. Each neuron may also contain rules which allow to send spikes (possibly with a delay) to other neurons, or to remove a given number of spikes from it (*firing* and *forgetting* rules).

Firing rules allow a neuron to send information to other neurons in the form of electrical impulses which are accumulated at the target cell. Forgetting rules remove from the neuron a predefined number of spikes. The application of every rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use one of its rules, then one of such rules must be used. If two or more rules could be applied, then only one of them is nondeterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons work in parallel with each other. A global clock is assumed, marking the time for the whole system, and hence the functioning of the system is synchronized.

From the seminal paper [16], other biological features have been explored in the framework of SN P systems. One of such extensions (with mathematical motivation) was introduced in [2], where a neuron can emit more than one spike, if the number of emitted ones is not greater than the consumed ones. Other variants including astrocytes [20], weights, which modify the number of spikes that arrives to a neuron according to the *quality* of the link between neurons [14, 21, 34], antispikes [22], or neuron division [33] have also been considered. In this paper, we will consider SN P systems with weights in the synapses and rules without delay[3].

In this way, the restricted model of SN P systems used in this paper can be formally described as follows. A *spiking neural P system* of degree $m \geq 1$ is a construct of the form

[3] For a more general description, see [16].

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \dots, \sigma_m$ 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) *Firing* rules $E/a^c \to a^d$, where E is a regular expression[4] over a, and $c \geq d \geq 1$ are integer numbers; if $E = a^b$ (with b an integer number, $b \geq c$), then the rule is usually written in the following simplified form: $a^b/a^c \to a^d$;
      (2) *Forgetting* rules $a^b/a^c \to \lambda$, for b and c integer numbers with $b \geq c \geq 1$.
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\} \times \mathbb{W}$ is the set of synapses between neurons, where the set of weights is $\mathbb{W} = \mathbb{Q} \cap [0, 1]$, i.e., the set of rational numbers between 0 and 1. The synapses also verifies that $(i, i, k) \notin syn$ for $1 \leq i \leq m$, $k \in \mathbb{W}$.
4. in is the label of the *input* neuron of Π.

A firing rule $E/a^c \to a^d \in R_i$ can be applied in neuron σ_i if it contains b spikes, $b \geq c$, and a^b belongs to the language associated to E. For applying rules of type $a^b/a^c \to a^d$, the neuron must contain exactly b spikes. The execution of these rules removes c spikes from σ_i (thus leaving the remaining spikes in σ_i), and sends d spikes to all the neurons σ_j such that $(i, j, k) \in syn$. The number of spikes that arrives to the neuron j through the synapse (i, j, k) depends on the number d of emitted spikes and the quality of the synapse, encoded by the *weight* k. In each neuron, the number of spikes after a computation step is the number of non consumed spikes plus the contribution of other neurons via the corresponding synapses. Let us consider that d spikes are emitted through a synapse (i, j, k). The contribution of σ_i to σ_j is an increase of $\lfloor d \times k \rfloor$ spikes, where $\lfloor v \rfloor$ denotes the largest integer not greater than v. A *forgetting* rule $a^b/a^c \to \lambda$ can be applied in neuron σ_i if it contains *exactly* b spikes. The execution of this rule simply removes all the c spikes from σ_i (thus leaving $b - c$ spikes).

A *configuration* of the system is described by the numbers $\langle n_1, n_2, \dots, n_m \rangle$ of spikes present in each neuron. At the beginning of the computation, the number of spikes in each neuron σ_i is n_i, but in the input neuron (with label in): If the input of the computation is N, then, in the initial configuration, the number of spikes in the input neuron is $n_{in} + N$.

Example 1. Let us consider the SN P system $\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn, 1)$ (see Fig. 1) with $\sigma_1 = (7, R_1)$, $\sigma_2 = (3, R_2)$, $\sigma_3 = (8, R_3)$ and the set of rules and synapses

[4] Along this paper, we use regular expressions of type $a^n a^*$, with $n \in \mathbb{N}$. In this case, the language associated to $a^n a^*$ is the set $\{a^{n+k} \mid k \in \mathbb{N}\}$. For more details about regular expressions, see, for example [29].

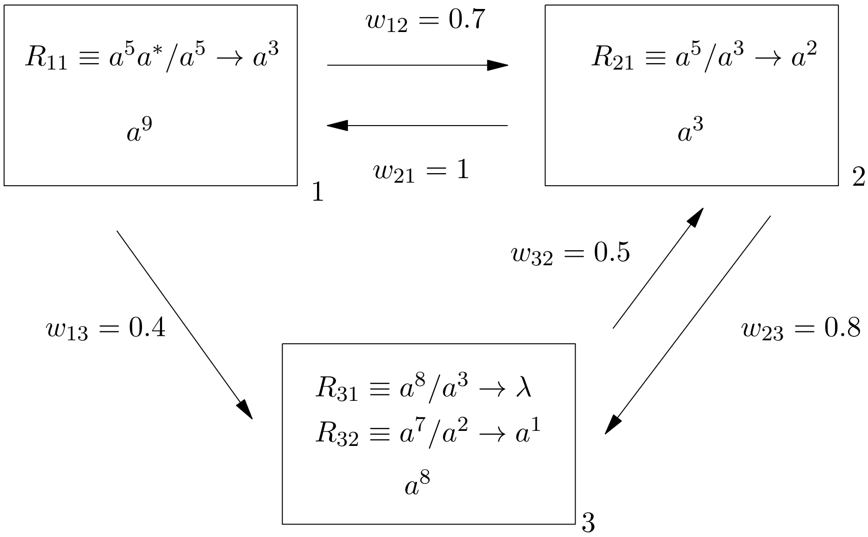

$R_{11} \equiv a^5a^*/a^5 \rightarrow a^3$
a^9
1
$w_{12} = 0.7$
$w_{21} = 1$
$R_{21} \equiv a^5/a^3 \rightarrow a^2$
a^3
2
$w_{32} = 0.5$
$w_{13} = 0.4$
$w_{23} = 0.8$
$R_{31} \equiv a^8/a^3 \rightarrow \lambda$
$R_{32} \equiv a^7/a^2 \rightarrow a^1$
a^8
3

Fig. 1. Example. The input neuron 1 has 9 = 7+2 spikes at the starting configuration.

$$R_1 = \{R_{11} \equiv a^5a^*/a^5 \rightarrow a^3\}$$
$$R_2 = \{R_{21} \equiv a^5/a^3 \rightarrow a^2\}$$
$$R_3 = \{R_{31} \equiv a^8/a^3 \rightarrow \lambda \quad , \quad R_{32} \equiv a^7/a^2 \rightarrow a^1\}$$

$$syn = \{(1,2,0.7),(2,1,1),(2,3,0.8),(3,2,0.5),(1,3,0.4)\}$$

We will consider the input $N = 2$, so, in order to start the computation, 9 spikes $(7 + 2)$ are placed in the neuron 1. Notice that R_{11} will be applied if the neuron 1 contains at least 5 spikes. R_{31} is the unique forgetting rule in the SN P system.

Since the input is $N = 2$, the initial configuration is $C_0 = \langle 9, 3, 8\rangle$. From this initial configuration, rules R_{11} and R_{31} can be applied. The rule R_{11} consumes 5 spikes from the neuron 1 and sends 3 spikes to the neurons 2 and 3. These 3 sent spikes are multiplied by the corresponding weights before arriving to the target neurons. The weight of the synapses between the neurons 1 and 2 is $w_{12} = 0.7$, so the application of the rule R_{11} produces an increase of $\lfloor 3 \times 0.7\rfloor = 2$ spikes in the neuron 2. Bearing in mind that $w_{13} = 0.4$, the application of the rule R_{11} increases in $\lfloor 3 \times 0.4\rfloor = 1$ the number of spikes in the neuron 3. The forgetting rule R_{31} deletes 3 spikes from neuron 3 and hence, the new obtained configuration is $C_1 = \langle 4, 5, 6\rangle$.

Now, the unique applicable rule is R_{21}. This rule consumes 3 spikes from neuron 2 and sends 2 spikes to the neurons 1 and 3. According with the corresponding weights, the number of the spikes in the neuron 1 is increased in $\lfloor 2 \times 1\rfloor = 2$ and the number of the spikes in the neuron 3 is increased in $\lfloor 2 \times 0.8\rfloor = 1$. By applying these modifications, the obtained configuration is $C_2 = \langle 6, 2, 7\rangle$.

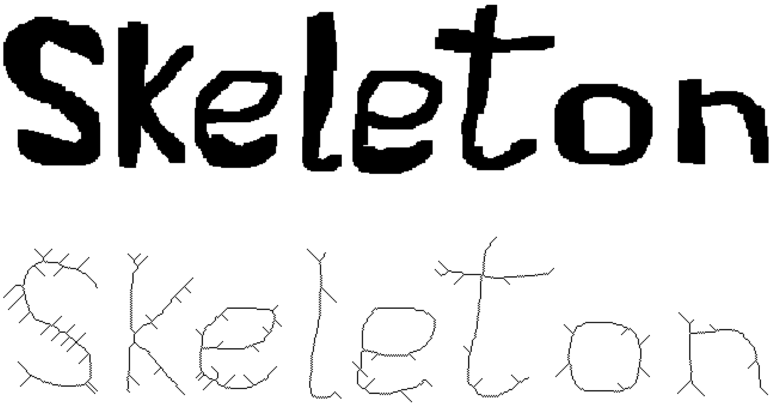


Fig. 2. A hand-written word and its skeletonization

From this configuration, rules R_{11} and R_{32} are applicable. The effects of R_{11} have been described above. The rule R_{32} removes 2 spikes from the neuron 3 and sends 1 spike to neuron 2. This spike is multiplied by the corresponding weight, $w_{32} = 0.5$, so it does not produce any increase in the number of spikes in neuron 3, since $\lfloor 0.5 \times 1 \rfloor = 0$. With these changes, the obtained configuration is $C_3 = \langle 1, 4, 4 \rangle$. No more rules can be applied and C_3 is the halting configuration.

3 Guo & Hall Algorithm

Skeletonization is a common transformation in Image Analysis. The concept of skeleton was introduced by Blum in [1], under the name of medial axis transform. There are many different definitions of the skeleton of a black and white image and many skeletonizing algorithms[5], but in general, the image B is a skeleton of the image A, if it has fewer black pixels than A, preserves its topological properties and, in some sense, keeps its *meaning.* In this paper, we focus on an iterative procedure of thinning: roughly speaking, the *border* black pixels are removed as long as they are not considered *significant.* The remaining set of black pixels is called the *skeleton* (See Fig. 2).

Among the parallel algorithms, special attention deserves the so-called 1-subcycle parallel algorithms or fully parallel algorithms [12]. Our bio-inspired design is based on a classical skeletonizing algorithm, the Guo & Hall algorithm [12, 13]. In this algorithm, the pixels are examined for deletion in an iterative process.

[5] A detailed description is out of the scope of this paper. For a survey in this topic, see e.g., [30].

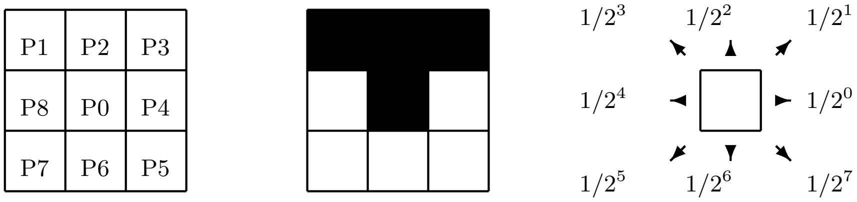


Fig. 3. (Left) Enumeration of the pixels in a 3×3 neighborhood. (Center) 3×3 neighborhood with encoding $[0, 0, 0, 0, 1, 1, 1, 1, 1]$, or, shortly, $2^4 + 2^5 + 2^6 + 2^7 + 2^8 = 496$. (Right) Scheme of the weights of the synapses.

First of all, given an $n \times m$ image, it is divided into two sub-sections. One of the sections is composed by the pixels a_{ij} such that $i + j$ is even. Alternatively, the second sub-section corresponds to the pixels a_{ij} such that $i + j$ is odd. The algorithm consists on two sub-iterations where the removal of redundant pixels from both sub-sections are alternated, i.e., in each step only the pixels of one of the subsections are evaluated for its deletion.

The decision is based on a 3×3 neighborhood. Given a pixel $P0$, a clockwise enumeration $P1, \ldots, P8$ of its eight neighbor pixels is considered, (Figure 3 (Left)). As usual, for each $i \in \{1, \ldots, 8\}$, Pi is considered as a Boolean variable, with the truth value 1 if Pi is *black* and 0 if Pi is *white*.

In order to decide if a pixel $P0$ is deleted in the corresponding iteration sub-cycle, two parameters are evaluated:

$$\begin{aligned} B(P0) &= \textstyle\sum_{i=1}^{i=8} Pi \\ C(P0) &= (\neg P2 \wedge (P3 \vee P4)) + (\neg P4 \wedge (P5 \vee P6)) \\ &\quad + (\neg P6 \wedge (P7 \vee P8)) + (\neg P8 \wedge (P1 \vee P2)) \end{aligned}$$

$B(P0)$ counts how many pixels in the neighborhood of $P0$ are black. $C(P0)$ evaluates the *connectivity* of the pixel $P0$. Notice that for isolated black pixels, the connectivity is 0, and for pixels surrounded by eight black pixels, the connectivity is 4.

According to the Guo & Hall algorithm, in each iteration, an evaluated black pixel $P0$ is deleted (changed to white) if and only if all of the following conditions are satisfied.

Guo & Hall conditions:

1. $B(P0) > 1$;
2. $C(P0) = 1$; This condition is necessary for preserving local connectivity when P is deleted.
3. $(P1 \wedge P3 \wedge P5 \wedge P7) \vee (P2 \wedge P4 \wedge P6 \wedge P8) = FALSE$; Intuitively, this condition is satisfied if $P0$ is not the central pixel of a *cross*.

For example, let us consider as $P0$ the central pixel in the image of Fig. 3 (Center). In this case, $B(P0) = 3 > 1$, $C(P0) = 1$, and the third condition is also satisfied. Hence, $P0$ will be deleted in the corresponding sub-cycle iteration.

4 SN P Systems for Skeletonizing

In this paper we will show how to use SN P systems for skeletonizing images. In particular, we use SN P systems for implementing the Guo & Hall algorithm. Without losing generality, we will consider each image as a mapping $I : \{1, \dots, p\} \times \{1, \dots, q\} \to \{black, white\}$, with $I(1, k) = I(p, k) = I(j, 1) = I(j, q) = white$ for all $k \in \{1, \dots, q\}$ and $j \in \{1, \dots, p\}$, i.e., the image has $n \times m$ pixels and all the pixels on the border are white.

Given a pixel (i, j), we can use the enumeration of the pixels used in the previous section to represent the neighborhood of the pixel $P0$ in (i, j). Such a neighborhood will be represented as a list $[H0, \dots, H8]$, where, for $r \in \{0, \dots, 8\}$, $Hr = 1$ if Pr is a white pixel and $Hr = 0$ if Pr is a black one[6]. This representation of the neighborhood can be done in a more compact way, by encoding the neighborhood as a number[7] in $\{0, \dots, 511\}$.

$$cod(i, j) = \sum_{r=0}^{8} Hr \times 2^r$$

For example, in Fig. 3 (Center), the 3×3 neighborhood can be encoded as $[0, 0, 0, 0, 1, 1, 1, 1, 1]$, or, shortly, $2^4 + 2^5 + 2^6 + 2^7 + 2^8 = 496$.

Since the decision of removing a black pixel (changing to white) depends on its 3×3 neighborhood and there is a bijective correspondence among the sets of all the possible neighborhoods and the possible encodings $\{0, \dots, 511\}$, it is easy to check that the pixel in (i, j) must be removed if it belongs to the set

$$DEL = \left\{ \begin{array}{rrrrrrrrrrrr} 6 & 12 & 14 & 18 & 24 & 26 & 28 & 30 & 36 & 38 & 44 & 46 \\ 48 & 50 & 56 & 58 & 60 & 62 & 66 & 72 & 74 & 96 & 98 & 104 \\ 106 & 112 & 114 & 120 & 122 & 124 & 126 & 132 & 134 & 140 & 142 & 144 \\ 146 & 152 & 154 & 156 & 158 & 164 & 166 & 172 & 174 & 176 & 178 & 184 \\ 186 & 188 & 190 & 192 & 194 & 200 & 202 & 224 & 226 & 232 & 234 & 240 \\ 242 & 248 & 250 & 252 & 258 & 262 & 264 & 266 & 270 & 286 & 288 & 290 \\ 294 & 296 & 298 & 302 & 318 & 384 & 386 & 390 & 392 & 394 & 398 & 414 \\ 416 & 418 & 422 & 424 & 426 & 430 & 448 & 450 & 454 & 456 & 458 & 462 \\ 480 & 482 & 486 & 488 & 490 & 496 & 498 & 504 & & & & \end{array} \right\}$$

The complementary set of encodings will be denoted by $\overline{DEL}$, in other words,

[6] Notice that we encode black pixels as 0 and white pixels as 1 for an easier implementation with SN P systems. In the previous section, we keep the opposite encoding in order to keep continuity with the literature.

[7] Similar ideas are also used in [30].

$$\overline{DEL} = \{s \in \{0, \ldots, 511\} \mid s \notin DEL\}$$

For each image of size $p \times q$, a SN P system will be provided. The input of the SN P system is a non negative integer which represents the number of iterations in the skeletonizing process. Formally, given an $p \times q$ image, we associate to it the following SN P system is degree $(p \times q) + 2$:

$$\Pi = (O, \sigma_{11}, \sigma_{12}, \ldots, \sigma_{pq}, \sigma_{odd}, \sigma_{even}, syn, odd),$$

i.e., a SN P system with a neuron for each pixel in the image plus two extra neurons, σ_{odd} and σ_{even}. The input neuron is σ_{odd}.

- $O = \{a\}$ is the singleton alphabet;
- $\sigma_{odd} = (512, a^{513}a^*/a^{513} \to a^{512})$ and $\sigma_{even} = (0, a^{512}/a^{512} \to a^{512})$.
- $\sigma_{ij} = (n_{ij}, R_{ij})$, $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, q\}$, where

$$n_{i,j} = \begin{cases} 0 & \text{if } i = 1 \vee i = p \vee j = 1 \vee j = q \\ cod(i,j) & \text{otherwise} \end{cases}$$

  $R_{1k} = R_{pk} = R_{j1} = R_{jq} = \emptyset$ for all $k \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, p\}$. For the remaining (i, j),

$$\begin{aligned} R_{ij} = \{&a^b/a^{511} \to a^{256} \mid b = r + 512 \wedge r \in DEL\} \cup \\ \{&a^b/a^{512} \to \lambda \mid b = r + 512 \wedge r \in \overline{DEL}\}; \end{aligned}$$

- $syn = \left(\bigcup_{i=2}^{p-1} \bigcup_{j=2}^{q-1} syn_{ij} \right)$
  $\cup\ syn_{odd} \cup syn_{even} \cup \{\langle odd, even, 1\rangle, \langle even, odd, 1\rangle\}$, where

$$syn_{odd} = \left\{ \begin{array}{ll} \langle odd, (i,j), 1\rangle \mid & i \in \{2, \ldots, p-1\}, j \in \{2, \ldots, q-1\}, \\ & i + j \ odd \end{array} \right\}$$

$$syn_{even} = \left\{ \begin{array}{ll} \langle even, (i,j), 1\rangle \mid & i \in \{2, \ldots, p-1\}, j \in \{2, \ldots, q-1\}, \\ & i + j \ even \end{array} \right\}$$

  and for all $i \in \{2, \ldots, q-1\}, j \in \{2, \ldots, q-1\}$,

$$syn_{ij} = \left\{ \begin{array}{ll} \langle (i,j), (i+1,j), 1/2^0\rangle, & \langle (i,j), (i-1,j), 1/2^4\rangle, \\ \langle (i,j), (i+1,j+1), 1/2^1\rangle, & \langle (i,j), (i-1,j-1), 1/2^5\rangle, \\ \langle (i,j), (i,j+1), 1/2^2\rangle, & \langle (i,j), (i,j-1), 1/2^6\rangle, \\ \langle (i,j), (i-1,j+1), 1/2^3\rangle, & \langle (i,j), (i+1,j-1), 1/2^7\rangle\} \end{array} \right\}$$

The SN P system has one neuron σ_{ij} for each pixel of the image plus two extra neurons σ_{odd} and σ_{even}. The neurons corresponding to the border of the image has zero spikes in the initial configuration, the remaining neurons σ_{ij} corresponding to the pixels of the image (called hereafter, the *regular neurons*) have $cod(i, j)$ spikes in the initial configuration. The neurons σ_{odd} and σ_{even} have 512 and 0 spikes respectively. We will add to the 512 spikes in σ_{odd} as many spikes as indicated as *input* for starting the computation.

The neuron σ_{odd} has only one rule $a^{513}a^*/a^{513} \to a^{512}$ which is applied if the neuron has at least 513 spikes. The neuron σ_{even} has also one rule $a^{512}/a^{512} \to a^{512}$. The regular neurons have two types of rules: *Firing* and *forgetting* ones, which will be applied if the number of spikes is exactly $b = r+512$ with $r \in DEL$ or $r \in \overline{DEL}$, respectively.

With respect to the synapses, a regular neuron corresponding to a pixel P is linked to the eight neurons corresponding to the eight neighbor pixels of P, with the weights $1/2^i$ where $i \in \{0, \ldots, 7\}$ follows an anti-clockwise enumeration of the pixels starting in the *east* pixel (see Fig. 3 (Right)). The neurons σ_{odd} and σ_{even} are linked each other. The neuron σ_{odd} is also linked to all the regular neurons σ_{ij} with $i+j$ odd and, analogously, σ_{even} is linked to all the regular neurons σ_{ij} with $i+j$ even.

4.1 How it works

In order to understand how the SN P system works, firstly we observe that at the initial configuration, the set of regular neurons σ_{ij} encodes the image which will be skeletonized. We will show that, at any time, these neurons encode the successive images obtained in the iterative process of deleting black pixels according to the Guo & Hall algorithm. Let us remark that if the number of spikes in σ_{ij} is even, then the corresponding pixel is black; otherwise, if the number of spikes is odd, then the corresponding pixel is white. This is easily derived from the definition of $cod(i,j)$.

Another observation to be considered is that the parity of the number of spikes in a neuron never changes, since the number of spikes received or removed is always an even amount, except by the application of the rule $a^b/a^{511} \to a^{256}$. As we will see below, the application of this rule is interpreted as the deletion of the corresponding black pixel in the Guo & Hall algorithm and it is applied once at most in each neuron.

Before explaining the different steps of the process, let us consider a pixel (i,j) in the image and a black pixel adjacent to (i,j). We identify this black pixel to $v \in \{P1, \ldots, P8\}$ according to the clockwise enumeration described above. Let us suppose that the black pixel in v belongs to the selected subsection in the current step of the Guo & Hall algorithm and it satisfies the conditions to be deleted.

Let us consider now the neurons σ_{ij} and σ_v corresponding to the pixels in (i,j) and v. As we will show below, the three conditions for v (it is black, it belongs to a selected subsection and it satisfies the Guo & Hall conditions to be deleted) indicates that the number of spikes in σ_v is $b = r + 512$ with $r \in DEL$. In this case the rule $a^b/a^{511} \to a^{256}$ is applied. As pointed out above, the application of this rule changes the parity of the number of spikes σ_r (from even, since the pixel is black, to odd) and this change is interpreted as a deletion of the pixel in v.

We focus on the influence of the deletion of the black pixel in v (or, equivalently, the application of the rule $a^b/a^{511} \to a^{256}$ in σ_v) on the neuron σ_{ij}. Since the number of spikes in σ_{ij} at the initial configuration is $cod(i,j) = \sum_{i=0}^{8} Hi \times 2^i$ and, in this configuration, the pixel v is black, according to the encoding, this means that Hv is zero, or, in other words, 2^v does nor appear in the encoding of the environment as an addition of powers of 2.

The deletion of the pixel in v changes the environment of (i,j) and then, since the number of spikes in σ_{ij} represents such environment, the number of spikes must change. In particular, the change corresponds to turn Hv to 1, or, in other words, to add 2^v to the number of spikes in σ_{ij}.

In order to check that this happens, it is suffices to seen that the rule $a^b/a^{511} \to a^{256}$ sends $256 = 2^8$ from neuron σ_v to σ_{ij}, but these 2^8 must be multiplied by the corresponding weight in $\{1/2^0, \ldots, 1/2^7\}$, so only 2, 2^2, 2^3,..., 2^7 or 2^8 spikes arrive to σ_{ij}, depending on the value of v. A simple inspection shows that the number of spikes that arrives to σ_{ij} is exactly 2^v when the black pixel v is deleted.

Bearing in mind these considerations, we show that for an input $N \in \{1, \ldots, 513\}$, the computation steps of the SN P system correspond to the iterative process of the Guo & Hall algorithm where the first selected subsection corresponds to pixels with $i + j$ odd. In such way we will show the following statements:

- **Statement 1:** The set of regular neurons is split into two sub-sections. One of the sections is composed by the neurons σ_{ij} such that $i+j$ is even. Alternatively, the second sub-section corresponds to the neurons σ_{ij} such that $i + j$ is odd. Both subsections are alternatively selected, starting with the *odd* subsection.
- **Statement 2:** In each computation step, only neurons corresponding to the selected subsection are evaluated. The evaluation consists on determining if the neighborhood of the pixel associated to the neuron satisfies the conditions of the Guo & Hall algorithm to be deleted.
- **Statement 3:** If an evaluated black pixel satisfies the Guo & Hall conditions to be deleted (see Section 3), then the number of spikes in the corresponding neuron changes from *even* to *odd*.
- **Statement 4:** In each configuration, the number of spikes in the regular neurons is the codification of an image, according to the encoding described above.

The first key point of the algorithm is that the image is split into two subsections which will be explored alternatively. One black pixel will be considered for its deletion only if it belongs to the subsection selected in the current step. We consider that a regular neuron σ_{ij} is selected at the step r if its number of spikes in the configuration C_r is greater than or equal to 512. Otherwise, if its number of spikes is lower than 512, then the neuron is not selected.

Next we show that the regular neurons with $i+j$ odd and even are alternatively selected. The selection of subsections is performed by the neurons σ_{odd} and σ_{even} which send, alternatively, 512 spikes to the regular neurons σ_{ij} with $i+j$ odd and even, respectively.

Lemma 1. Let σ_{ij} be a regular neuron and $N \in \{1, \ldots, 513\}$ the input of the SN P system. For $r \in \{0, \ldots, N-1\}$

- If $i + j$ is odd, then the number of spikes in σ_{ij} is greater than or equal to 512 in the configuration C_{2r+1} and it is lower than 512 in the configuration C_{2r}.
- If $i + j$ is even, then the number of spikes in σ_{ij} is greater than or equal to belongs to 512 in the configuration C_{2r+2} and it is lower than 511 in the configuration C_{2r+1}.

Proof. Let us observe that in the initial configuration, the number of spikes in a regular neuron σ_{ij} is $cod(i, j) < 512$; the number of spikes in σ_{odd} is $512 + N$ and there is zero spikes in the neuron σ_{even}. From this initial configuration, the unique applicable rule is $a^{513}a^*/a^{513} \to a^{512}$ in the neuron σ_{odd} (since $N \geq 1$). After applying this rule, in the configuration C_1, the number of spikes in σ_{odd} is $N - 1$; the number of spikes in σ_{even} is 512; and the number of spikes in σ_{ij} is $cod(i, j) + 512$ if $i + j$ is odd and $cod(i, j)$ if $i + j$ is even.

Let us focus now on σ_{odd} and σ_{even}. The unique neuron that sends spikes to σ_{odd} is σ_{even} and, analogously, the unique neuron that sends spikes to σ_{even} is σ_{odd}. In the configuration C_1, the spikes in σ_{odd} and σ_{even} are $N - 1$ and 512, respectively. Since $N \leq 513$, the rule in σ_{odd} cannot be applied in this configuration, but the rule in σ_{even} can be applied, so the spikes in σ_{odd} and σ_{even} in the configuration C_2 are $512 + N - 1$ and 0, which is similar to the situation in the initial configuration, so we have that for $r \in \{1, \ldots, N\}$, the number of spikes in σ_{odd} and σ_{even} and in the configuration C_{2r} are $512 + N - r$ and 0.

Notice that at the configuration C_{2N}, the number of spikes in σ_{odd} and σ_{even} are 512 and 0, respectively, and no more rules are applied in these neurons.

According to the number of spikes in σ_{odd} and σ_{even} in the odd and even configurations, and taken into account their synapses, then we have that, for $r \in \{0, \ldots, N - 1\}$, at the configuration C_{2r+1}, the regular neurons with $i + j$ odd has at least 512 spikes; and at C_{2r+2}, the regular neurons σ_{ij} with $i + j$ even has at least 512 spikes.

In order to complete the proof, it is necessary to prove that for $r \in \{0, \ldots, N - 1\}$, at the configuration C_{2r+1}, the regular neurons with $i + j$ even has at most 511 spikes; and at C_{2r+2}, the regular neurons σ_{ij} with $i + j$ odd has at most 511 spikes.

Let us start by considering a regular neuron σ_{ij} with $i + j$ odd at the configuration C_1. As we show above, its number of spikes is $b = 512 + r$ with $r = cod(i, j) \leq 511$. Depending on $r \in DEL$ or $r \in \overline{DEL}$, one of the rules $a^b/a^{511} \to a^{256}$ or $a^b/a^{512} \to \lambda$ is applied.

- Let us suppose that $r \in DEL$. In particular, this means that the corresponding pixel is black and the applied rule is $a^b/a^{511} \to a^{256}$. The number of spikes in the configuration C_2 is equal to the spikes in the configuration C_1 $(512 + r)$, minus the consumed ones 511 plus the contribution of other neurons. Since σ_{odd} does not send any spike in this step, the unique contribution to the number of spikes comes from other regular neurons. Each contribution is the addition of 2^i spikes to the spikes in σ_{ij}, but, bearing in mind that the pixel is black, then 2^0 does not appears in the decomposition of $cod(i, j)$ as sum of powers of 2, and it cannot be added as a contribution of other neuron, so r plus the contribution of other neurons is at most 510 and then, the number of spikes in σ_{ij} in C_2 is lower than 512.

- If $r \notin DEL$, then the rule $a^b/a^{512} \to \lambda$ is applied. Since 512 spikes are consumed and r plus the contributions of the other regular neurons is at most 511, then the number of spikes in σ_{ij} is lower than 512, also in this case.

This reasoning can be also applied to show that the number of spikes of the neurons σ_{ij} with $i+j$ even is lower than 512 in the configuration C_3 and in general we have that for $r \in \{0, \ldots, N-1\}$, at the configuration C_{2r+1}, the regular neurons with $i+j$ even has at most 511 spikes; and at C_{2r+2}, the regular neurons σ_{ij} with $i+j$ odd has at most 511 spikes. □

The previous lemma shows that the property *to have at least 512 spikes* changes alternatively from neurons σ_{ij} with $i+j$ odd and even. From this result, it is easy to check the second statement, since the rules in the regular neurons can only be applied if the number of spikes is at least 512. That means that only in such cases the neuron is considered for evaluation.

Evaluating a neuron consists on deciding if the rule $a^b/a^{511} \to a^{256}$ or $a^b/a^{512} \to \lambda$ is applied, but such decision depends on the set DEL which are the set of encodings of the neighborhood such that the central pixel must be deleted.

The next key point of the algorithm is the deletion of pixels. By definition of $cod(i,j)$, a regular neuron σ_{ij} has an odd number of spikes if and only if it represents a white pixel. Analogously, a regular neuron σ_{ij} has an odd number of spikes if and only if it represents a white pixel. Bearing in mind this coding of black pixels, deleting a black pixel in a computation step consists on removing an odd amount of spikes from a neuron with an even amount of spikes.

Lemma 2. Let us consider a black pixel and a step of the Guo & Hall algorithm, such that the pixel belongs to the selected subsection and it satisfies the conditions of the algorithm to be deleted. Then, in the corresponding step of the SN P system computation, the corresponding regular neuron σ_{ij} will pass from an odd amount of spikes to an even amount.

Proof. According to the previous construction, a black pixel which belongs to the selected subsection in the Guo & Hall algorithm and verifies the conditions to be deleted has associated a regular neuron with $512 + r$ spikes, $r \in DEL$. In this case, the rule $a^b/a^{511} \to a^{256}$ is applied. Bearing in mind that r is even, the contributions of other neurons is even and an odd number of spikes is consumed, in the next configuration, the number spikes in the neuron is odd. □

Since all the $r \in DEL$ are even and the contribution of other neurons is always even, then the rule $a^b/a^{511} \to a^{256}$ with $b = 512 + r$ is applied at most once in each neuron. This means that if a pixel is deleted (changed from black to white) it never becomes black to white, and the iterative process of thinning the image is also carried out in the SN P system.

Finally, to sum up these statements. We claim the following result.

Theorem. The set of regular neurons of the SN P system encodes in each configuration the successive images obtained in the iterative process of thinning of the Guo & Hall algorithm, by taking as *black* the pixels with an even amount of spikes and *white* the neurons with an odd amount of spikes.

5 Experimental Simulation

Simulation of different variants of P systems have been widely studied in the last years. Since there do not exist implementations of P systems *in vivo* nor *in vitro*, the natural way to explore the behavior of designed P systems is to simulate it in conventional computers. A short description of some of these simulators can be found in [7, 15]. Currently, a big effort is being developed in the *P-lingua project* [8], by combining an efficient simulation engine with an *ad-hoc* programming language.

In this paper, a software tools based on the design of the SN P system has been implemented by using CUDA$^{\text{TM}}$, (Compute Unified Device Architecture) [18, 19]. CUDA$^{\text{TM}}$ is a general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processors Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU.

The experiments have been performed on a computer with a CPU AMD Athlon II x4 645, which allows to work with four cores of 64 bits to 3.1 GHz. The computer has four blocks of 512KB of L2 cache memory and 4 GB DDR3 to 1600 MHz of main memory.

The used graphical card (GPU) is an NVIDIA Geforce GT240 composed by 12 *Stream Processors* with a total of 96 cores to 1340 MHz. It has 1 GB DDR3 main memory in a 128 bits bus to 700 MHz. So, the transfer rate obtained is by 54.4

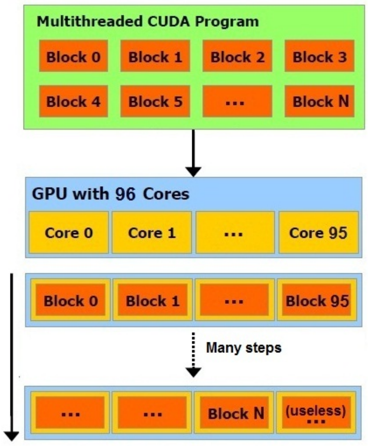


Fig. 4. Scheme of the threads

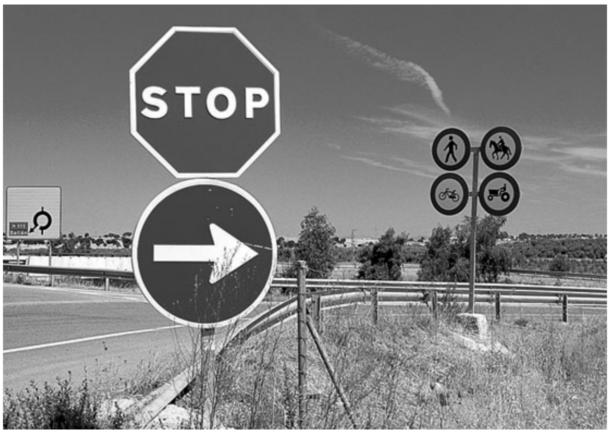


Fig. 5. Example of image with traffic signals

Gbps. The used Constant Memory is 64 KB and the Shared Memory is 16 KB. Its Compute Capability level is 1.2 (from 1.0 to 2.1). The implementation deals with N blocks of threads for the complete image in our GPU of 96 cores, as we can see in Fig. 4. We need more threads than pixels if the height and width of the image are not multiples of 16; i.e., we can have useless threads (see Figure 4).

5.1 Examples

A first example is shown in Fig 2. Skeletonizing hand-written texts is one of the challenges of skeletonizing, since the skeleton keeps the topological structure and meaning of the original and the text can be easily stored.

Next, we provide several examples of a realistic recognizing problem with applications in the automotive industry. In Fig. 5, we can see a photograph taken in a road. It has been binarized by using a threshold method by using a threshold 100 on a gray scale $0, \ldots, 255$. We can see that the skeletonized images keep the information of the traffic signals and they can be used in a further pattern recognition problem (see Fig. 6). In Fig. 7, two more examples of skeletionizing real images are shown.

We finish this section by showing the results of some experiments performed with our implementation. We have taken 36 totally black images of $n \times n$ pixels[8], from $n = 125$ to $n = 4500$ with a regular increment of 125 pixels of side. Figure 8 (top) shows the time in milliseconds of our software tool inspired in the designed SN P system for implementing the Guo & Hall algorithm for 1, 30, 60 and 90 steps in the skeletonizing process. Figure 8 (bottom) shows the same study for a sequential implementation of the algorithm.

[8] Theoretically, this is the worst case, since the time inverted by the algorithm depends on the size of the biggest black connected component of the original image.

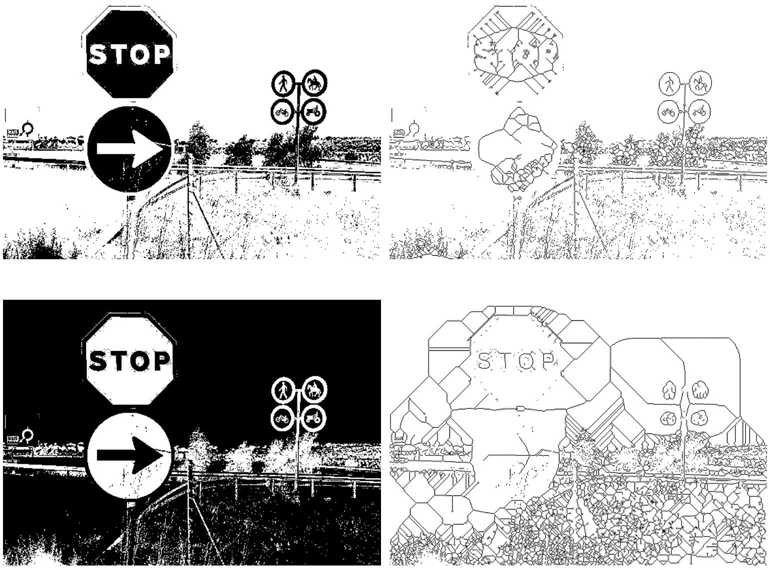


Fig. 6. (Top left) The binarization of the image from Figure 5 (Top right) Inverse binarization of the image. (Bottom left) Its skeletonizing. (Bottom right) The skeletonizing of the inverse thresholding.

6 Conclusions

The development of new bioinspired parallel techniques provides a chance for revisiting classical sequential algorithms. In this paper, we have consider a classical algorithm for skeletonizing images, but many other algorithms can be considered. In particular, the bio-inspired computing techniques have features as the encapsulation of the information, a simple representation of the knowledge and parallelism, which are appropriate with dealing with digital images.

Nonetheless, the use of new computational paradigms for developing the bio-inspired ideas needs, on the one hand, the contribution of theoretical research that allows us to design new bio-inspired efficient algorithms, and, on the other hand, the use of the most recent parallel computer architectures for a real parallel implementation of the algorithms.

In this paper we provide a new step in both directions, since we study the skeletonization of images by using Spiking Neural P systems and show the results of a new software based on the SN P system by using the GPU architecture. This research line can be followed by considering more classical problems and studying the possible improvements from a bio-inspired perspective, or, by studying the same skeletonization problem in other P system models.

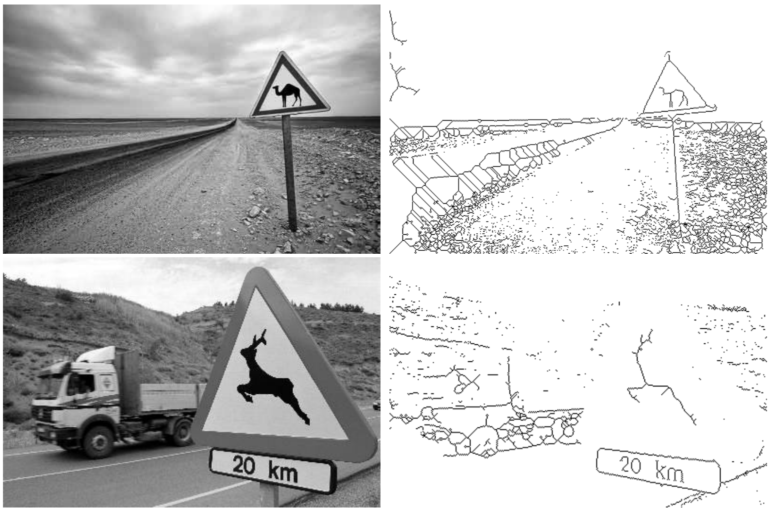


Fig. 7. Original images and their skeletons

Acknowledgements

DDP and MAGN 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. Blum, H.: An associative machine for dealing with the visual field and some of its biological implications. In: Bernard, E.E., Kare, M.R. (eds.) Biological Prototypes and Synthetic Systems. vol. 1, pp. 244–260. Plenum Press, New York (1962)
2. Chen, H., Ionescu, M., Ishdorj, T.O., Păun, A., Păun, Gh., Péz-Jiméz, M.: Spiking neural P systems with extended rules: universality and languages. Natural Computing 7, 147–166 (2008)
3. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Thresholding of 2D images with cell-like P systems. Romanian Journal of Information Science and Technology 13(2), 131–140 (2010)
4. Christinal, H.A., Díaz-Pernil, D., Real, P.: Segmentation in 2D and 3D image using tissue-like P system. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, CIARP 2009. Lecture Notes in Computer Science, vol. 5856, pp. 169–176. Springer (2009)
5. Christinal, H.A., Díaz-Pernil, D., Real, P.: Region-based segmentation of 2D and 3D images with tissue-like P systems. Pattern Recognition Letters 32(16), 2206 – 2212 (2011)

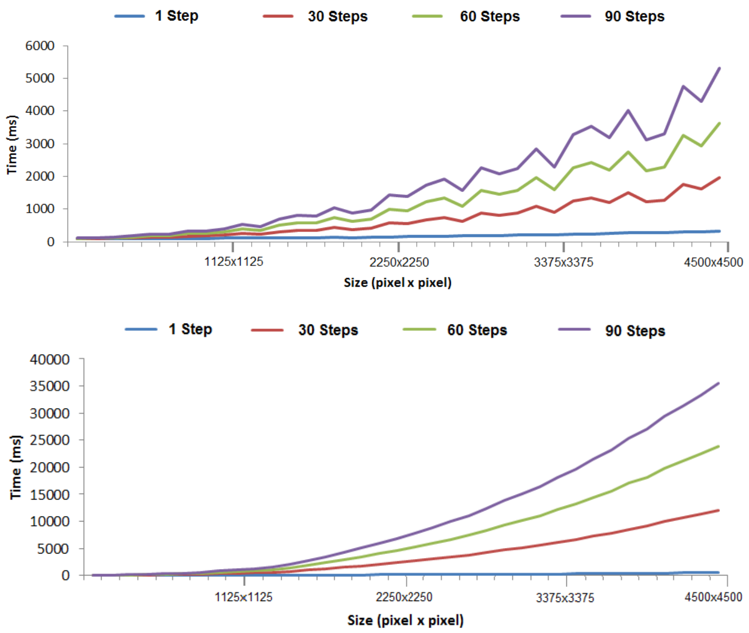


Fig. 8. Experimental time obtained for the Guo & Hall algorithm 36 totally black images of $n \times n$ pixels, from $n = 125$ to $n = 4500$ with a regular increment of 125 pixels of side. Top image shows the time of our parallel implementation in SN P Systems. Bottom image shows the time for a sequential implementation.

6. Corne, D.W., Frisco, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol. 5391. Springer (2009)
7. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for P systems. In: Păun et al. [27], pp. 437–454
8. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-lingua programming environment for membrane computing. In: Corne et al. [6], pp. 187–203
9. Egmont-Petersen, M., de Ridder, D., Handels, H.: Image processing with neural networks - a review. Pattern Recognition 35(10), 2279–2301 (2002)
10. Gimel'farb, G., Nicolescu, R., Ragavan, S.: P systems in stereo matching. In: Real, P., Díaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W. (eds.) Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol. 6855, pp. 285–292. Springer Berlin / Heidelberg (2011)
11. Gimel'farb, G.L.: Probabilistic regularisation and symmetry in binocular dynamic programming stereo. Pattern Recognition Letters 23(4), 431–442 (2002)
12. Guo, Z., Hall, R.W.: Parallel thinning with two-subiteration algorithms. Communications of the ACM 32, 359–373 (1989)

13. Guo, Z., Hall, R.W.: Fast fully parallel thinning algorithms. CVGIP: Image Understanding. 55, 317–328 (1992)
14. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Hebbian learning from spiking neural P systems view. In: Corne et al. [6], pp. 217–230
15. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Available membrane computing software. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, Gh. (eds.) Applications of Membrane Computing, pp. 411–436. Natural Computing Series, Springer (2006)
16. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71(2-3), 279–308 (2006)
17. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
18. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue 6, 40–53 (2008)
19. Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C.: GPU Computing. Proceedings of the IEEE 96(5), 879–899 (2008)
20. Pan, L., Wang, J., Hoogeboom, H.: Spiking neural P systems with astrocytes. 24(3), 805–825 (2012)
21. Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural P systems with weighted synapses. Neural Processing Letters 35(1), 13–27 (2012)
22. Pan, L., Păun, Gh.: Spiking neural P systems with anti-spikes. International Journal of Computers, Communications & Control IV(3), 273–282 (September 2009)
23. Peña-Cantillana, F., Díaz-Pernil, D., Berciano, A., Gutiérrez-Naranjo, M.A.: A parallel implementation of the thresholding problem by using tissue-like P systems. In: Real, P., Díaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W.G. (eds.) CAIP (2). Lecture Notes in Computer Science, vol. 6855, pp. 277–284. Springer (2011)
24. Peña-Cantillana, F., Díaz-Pernil, D., Christinal, H.A., Gutiérrez-Naranjo, M.A.: Implementation on CUDA of the smoothing problem with tissue-like P systems. International Journal of Natural Computing Research 2(3), 25–34 (2011)
25. Păun, Gh.: Computing with membranes. Tech. Rep. 208, Turku Centre for Computer Science, Turku, Finland (November 1998)
26. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin, Germany (2002)
27. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)
28. Rosin, P.L.: Training cellular automata for image processing. IEEE Transactions on Image Processing 15(7), 2076–2087 (2006)
29. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages: Word, language, grammar. Handbook of Formal Languages, Springer (1997)
30. Saeed, K., Tabedzki, M., Rybnik, M., Adamski, M.: K3M: A universal algorithm for image skeletonization and a review of thinning techniques. Applied Mathematics and Computer Science 20(2), 317–335 (2010)
31. Selvapeter, P.J., Hordijk, W.: Cellular automata for image noise filtering. In: World Congress on Nature & Biologically Inspired Computing, 2009. NaBIC 2009. pp. 193–197. IEEE (2009)
32. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)

33. Wang, J., Hoogeboom, H.J., Pan, L.: Spiking neural P systems with neuron division. In: Gheorghe, M., Hinze, T., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Int. Conf. on Membrane Computing. Lecture Notes in Computer Science, vol. 6501, pp. 361–376. Springer, Berlin Heidelberg (2010)
34. Wang, J., Hoogeboom, H.J., Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with weights. Neural Computation 22(10), 2615–46 (2010)
35. Zhou, Y., Chellappa, R.: Artificial neural networks for computer vision. Research notes in neural computing, Springer-Verlag (1992)

A Formal Framework for P Systems with Dynamic Structure

Rudolf Freund[1], Ignacio Pérez-Hurtado[2],
Agustín Riscos-Núñez[2], Sergey Verlan[3]

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

[2] Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla,
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
Email: {perezh,ariscosn}@us.es

[3] LACL, Département Informatique, Université Paris Est,
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@univ-paris12.fr

Summary. This article introduces a formalism/framework able to describe different variants of P systems having a dynamic structure. This framework can be useful for the definition of new variants of P systems with dynamic structure, for the comparison of existing definitions as well as for their extension. We give a precise definition of the formalism and show how existing variants of P systems with dynamic structure can be translated to it.

1 Introduction

This article is an attempt to fulfill the goal of defining a formal framework that captures the essential properties of P systems with dynamic structure. This framework can be seen as a kind of meta-language that permits to describe a P system and its evolution. Our main goal is to provide a simple tool for the analysis of different models of P systems with dynamic structure. There are numerous possible applications of the results of such an analysis, as, for example, the comparison and the extension of existing models and the creation of new models of P systems with a dynamic structure.

The article extends the approach used in [3] for P systems with static structure. We recall that the framework for the static P systems is mainly composed of five ingredients: the definition of the configuration of the system, the definition of rules, the definition of the applicability and of the application of a rule/multiset of rules, transition mode and halting condition. The configuration is a list of multisets corresponding to the contents of membranes of a P system and the rules generalize most kind of rules used in the P systems area. Based on this general form of rules, the applicability and the application of a (group of) rule(s) are defined using an algorithm. This permits to compute the set of all applicable multisets of rules for a concrete configuration C ($Applicable(\Pi, C)$). Then this set is restricted according to the transition mode δ ($Applicable(\Pi, C, \delta)$). For the transition, one of the multisets from this last set is non-deterministically chosen and applied, yielding a new configuration. The result of the computation is collected when the system halts according to the halting condition, which corresponds to a predicate that depends on the configuration and the set of rules.

In the case of P systems with dynamic structure the first three ingredients are to be changed in order to accommodate with the fact that the structure of the system can change. Informally, a *configuration* is a list of triples (i, h, w), where i is the unique identifier of a cell/membrane, h is its label and w is its contents. A configuration also contains the description of the structure of the system, which is given by a binary relation ρ on cell identifiers.

We assume that the set of rules is fixed (does not change in time). Rule actions are expressed in terms of "virtual" cells (membranes). These virtual cells are identified by labels. The process of the application of rules first makes a correspondence between the current configuration and the virtual cells described in a rule, i.e. it tries to "match" the constraints of virtual cells (labels, relation, contents, etc.) against the current configuration. When a subset of cells from the current configuration (say $\mathcal{I}$) matches the constraints of a rule, we say that a rule can be *instantiated* by the instance $\mathcal{I}$. The instantiation of r by $\mathcal{I}$ is the couple $(r, \mathcal{I})$, denoted by $r\langle\mathcal{I}\rangle$, and it can then be treated as a rule that could be applied like in the static case. The rules also contain additional ingredients that permit to modify the structure (the relation ρ).

Instances of rules can further be used to compute the applicable set of multisets of rules and we provide an algorithm for this purpose. The transition modes and halting conditions can easily be applied to this set exactly as in the static case.

The article is organized as follows. Section 2 gives the definition of the framework and presents the related algorithms. Section 3 presents a taxonomy that permits to define shortcuts for the commonly used cases. Then in section 4 we give examples of the translation of different types of P systems with dynamical structure. Finally, we discuss the perspectives of the presented approach.

2 Definitions

We assume that the reader is familiar with standard definitions in formal language theory (for example, we refer to [8] for all details) and with standard notions of P systems, as described in the books [5] and [6] or at the web page [7].

2.1 Graph transformations

There exist several ways to define a graph transformation. We will define a the graph transducer using the formalism from [2]. This formalism defines the graph transformation as a graph-controlled graph rewriting grammar with appearance checking using the following operations:

- $I(X)$: creation of a new node labeled by X;
- $D(X)$: deletion of a node labeled by X;
- $C(X,Y)$: change the label of the node labeled by X to Y;
- $I(l_1,\lambda,l_2;l_1',a,l_2')$: insert an edge labeled by a between two nodes labeled by l_1 and l_2; after the insertion nodes are relabeled to l_1' and l_2' respectively;
- $D(l_1,a,l_2;l_1',\lambda,l_2')$: delete the edge labeled by a between two nodes labeled by l_1 and l_2; after the deletion nodes are relabeled to l_1' and l_2' respectively;
- $C(l_1,a,l_2;l_1',a',l_2')$: rename to a' the label of the edge labeled by a between two nodes labeled by l_1 and l_2, After this operation nodes are relabeled to l_1' and l_2' respectively.

It was proved in [2] that the above formalism is computationally complete.

In what follows we will use some particular graph transducers whose definition we give below:

- $DELETE(x)$: $C(x,x')$, $D(x',a,y;x',\lambda,y)$ (looping over a and y), $D(x')$
- $INSERT(x)$: $I(x)$
- $INSERT-EDGE(x,y)$: $I(x,\lambda,y;x,a,y)$
- $DELETE-EDGE(x,y)$: $D(x,a,y;x,\lambda,y)$

2.2 Definition of the framework

We start by defining a configuration of a P system. Since we deal with P systems with dynamic structure, it should be taken into account that the number of cells (membranes) is not fixed (it is unbounded) and it will be represented by a list.

Definition 1 *A* basic configuration *C (of size n) is a list $(i_1,w_1)\ldots(i_n,w_n)$, where each w_j is a multiset (over O) and each $i_j \in \mathbb{N}$, $i_j \neq i_k$, for $k \neq j$, $1 \leq j,k \leq n$.*

If not stated otherwise, we suppose that all multisets of a basic configuration are finite. If needed the definitions that follow can be adapted to infinite multisets by adding corresponding constraints to the rule definition like it was done in [3].

The set of basic configurations of any size $n > 0$ is denoted by $\mathbb{C}$. We remark that we will consider only basic configurations of finite size and we denote the size of C by size(C).

Each element (i_j,w_j), $1 \leq j \leq n$, of a configuration C is called a *cell*. We say that i_j is the *id* of the cell j and that w_j is the contents of the cell j. We define the function $id(x)$ which for a cell x returns its id. We require the function id to be bijective, i.e., there should be a one-to-one correspondence between cells and their id's.

If not stated otherwise, we will consider that id is the identity function ($id(x) = x$), so we will not distinguish between an id and its position in the list of configurations.

Definition 2 *A* configuration $\mathcal{C}$ *is the couple* (L, ρ)*, where* $L \in Lab \times \mathbb{C}$ *is the list* $(i_1, l_1, w_1) \dots (i_n, l_n, w_n)$ *with* (i_j, w_j) *corresponding to an element of a basic configuration and* $l_j \in Lab$ *being the* label *of the cell,* $1 \leq j \leq n$ *(Lab is a set of labels). The second component* $\rho \subseteq \mathbb{N} \times \mathbb{N}$ *is the relation graph between cells.*

Hence in a configuration any cell has an id (equal to the position) which is unique and a label which is not necessarily unique. We define the function $lab(x) : \mathbb{N} \to L$ that returns the label of the cell having the id equal to x. We denote by $\mathcal{C}_m$ and $\mathcal{C}_\rho$ the first and the second components of the configuration $\mathcal{C}$. We also denote by $\bar{\mathcal{C}_m} \in \mathbb{C}$ the projection of $\mathcal{C}_m$ erasing the labels (yielding a basic configuration).

The relation ρ is defined on id's of cells being part of the configuration. In P systems this corresponds to the parent relation, while in tissue P systems this corresponds to the communication graph of the system.

The set of all possible configurations is denoted by $\mathfrak{C}$.

Now we will give the definition of a rule. A rule r is defined by the following components. We remark that all of them are given in terms of virtual cell positions. We will also call them **relative** positions because they are introduced in the first component of the rule.

A. **Checking**

1. $Labels(r) \in Lab^*$ ($Labels(r) = (l_1, \dots, l_k)$) is a list of cell labels. This list identifies k (relative) positions labeled from 1 to k that we further call virtual cells. Let $\mathbb{N}_k = \{1, \dots, k\}$ and $\mathbb{K}$ be a subset of $\mathbb{C}$ where for any cell x it holds $1 \leq id(x) \leq k$.
2. $\rho(r) \subseteq \mathbb{N}_k \times \mathbb{N}_k$ is the constraint imposed by the (parent) relation on the virtual cells.
3. $Perm(r) \subseteq \mathbb{K}$ defines the permitting condition.
4. $For(r) \subseteq \mathbb{K}$ defines the forbidding condition.

B. **Modification of existing configuration/structure**

5. $Rewrite(r) \in (\mathbb{K} \times \mathbb{K})$ is a general rewriting rule permitting to rewrite a finite basic configuration to another one (e.g., $(j, u)(i, v) \to (m, w)$). By $Bound(r)$ we denote the first component (the left-hand-side) of this rewriting rule.
6. $Label\text{–}Rename(r) \in (\mathbb{N}_k \times Lab)^*$ renames the labels specified by the list.
7. $Delete(r) \in \mathbb{N}_k^*$ gives the indexes of virtual cells to be deleted.
8. $Delete\text{–}and\text{–}Move(r) \in (\mathbb{N}_k \times \mathbb{N}_k)^*$ is a list of couples of indices (e.g., (j, k)) indicates that the virtual cell j should be deleted and its contents should be moved to the virtual cell k).

C. **Creation of new structures**

9. $Generate(r) \in (\mathbb{N}' \times Lab \times O^{\circ})^*$ is a list of triples consisting of a (primed) index, a label, and a multiset (e.g. (j', h, u)). This component introduces new cells to be created by the application of the rule.
10. $Generate\text{–}and\text{–}Copy(r) \in (\mathbb{N}' \times Lab \times \mathbb{N} \times \bar{\mathcal{R}})$ -is a list of quadruplets consisting of a (primed) index, a label, an index, and a rewriting rule (e.g. $(j', h, i, u \to v)$). This component specifies new cells to be created by duplicating existing cells.
    We denote the smallest multiset containing any left-hand side of rewriting rules from *Generate–and–Copy* by $DPerm(r)$.

D. **Structure transformation**

11. *Change–Relation* is a graph transducer that updates the relation ρ. This transducer should be recursive and it can only add and remove edges (no node creation/removal is allowed).

Now we define what means the applicability of a rule. Before giving the algorithm, we define some additional notions related to relative positions.

An *instance* of size n is a vector of integers $I = (i_1, \dots, i_n)$, $i_j \in \mathbb{N}$, $1 \le j \le n$. By size(I) we denote the size of an instance I, and by $I|_k$, $1 \le k \le n$, the k-th value of the vector I, i.e., i_k.

For a basic configuration $C \in \mathbb{C}$, $C = (j_1, w_1) \dots (j_k, w_k)$, and for an instance I we define the *instantiation* of C by I, denoted $C\langle I\rangle$, as follows:

$$C\langle I\rangle = (I|_{j_1}, w_1), \dots, (I|_{j_k}, w_k).$$

In the above formula we assume that the cells of configuration C do not necessarily have their id in the range $[1 \dots \text{size}(C)]$. We also remark that size$(C) \le$ size(I).

It is clear that if C is defined in terms of relative positions then $C\langle I\rangle$ permits to replace these relative positions by the corresponding values from I (a relative position k is replaced by $I|_k$ which is i_k).

For a rule r as defined above and for an instance I such that $|Labels(r)| \le$ size(I) we obtain the instantiation of r by I, denoted by $r\langle I\rangle$, by replacing all relative positions k by $I|_k$ in $Perm(r)$, $For(r)$, $Rewrite(r)$, $Label\text{–}Rename(r)$, $Delete(r)$, $Delete\text{–}and\text{–}Move(r)$ and $Change\text{–}Relation(r)$.

Now we define what means the applicability of a group of rules. First we define the set of *valid instances* for a rule $r \in \mathbb{R}$ in a configuration $\mathcal{C}$. This set, denoted by $\mathcal{I}_{\mathcal{C}}(r)$, is obtained by the following algorithm.

1. Getting instances in conformity with $Labels(r)$:

$$\bar{\mathcal{I}}_{\mathcal{C}}(r) = \{(i_1, \dots, i_k) \mid (l_1, \dots, l_k) = Labels(r) \text{ and } lab(i_j) = l_j,\\ 1 \le i_j \le \text{size}(\mathcal{C}), 1 \le j \le k\}.$$

2. Checking the relation ρ:

$$\mathcal{I}_{\mathcal{C}}(r) = \{(i_1, \dots, i_k) \in \bar{\mathcal{I}}_{\mathcal{C}}(r) \mid (j, m) \in \rho(r) \Rightarrow (i_j, i_m) \in \mathcal{C}_\rho\}.$$

For a multiset of rules $R \in \mathbb{R}^\circ$ and a configuration $\mathcal{C} \in \mathfrak{C}$ we define the set of multisets *Applicable*$(R, \mathcal{C}) \subseteq (\mathbb{R} \times \mathbb{N}^*)^\circ$ giving the set of multisets of instantiated rules that can be computed based on R and the configuration $\mathcal{C}$. This set is computed as follows.

Let $R = \{r_1, \dots, r_n\}$ (the rules are not necessarily different) and let $\mathcal{I}_\mathcal{C}(r_i) = (v_{i,1}, \dots, v_{ik_i})$, $1 \leq i \leq n$. Consider an arbitrary vector of rule instances $v = (v_{1,j_1}, \dots, v_{n,j_n})$, $1 \leq j_i \leq k_i$, $1 \leq i \leq n$. The multiset $\{(r_1, v_{1,j_1}), \dots, (r_n, v_{n,j_n})\}$ is added to *Applicable*$(R, \mathcal{C})$ if

- For all $p \in Perm(r_i) \cup DPerm(r_i)$, $p\langle v\rangle \subseteq \bar{\mathcal{C}}_m$, $1 \leq i \leq n$.
- For all $q \in For(r_i)$, $q\langle v\rangle \not\subseteq \bar{\mathcal{C}}_m$, $1 \leq i \leq n$.
- $\bigcup_{i=1}^n Bound(r_i)\langle v\rangle \subseteq \bar{\mathcal{C}}_m$.
- The consecutive application of graph transducers *Change–Relation*(r_i) and *Change–Relation*(r_j) yields the same result regardless of the order of the application, $1 \leq i, j \leq n$.

It is clear that there is a bound on the size of the multiset of rules R for which *Applicable*$(R, \mathcal{C})$ is not empty. We denote by *Applicable*$(\mathcal{C}) \subseteq (\mathbb{R} \times \mathbb{N}^*)^\circ$ the union of corresponding multisets:

$$Applicable(\mathcal{C}) = \bigcup_{Applicable(R,\mathcal{C}) \neq \emptyset} Applicable(R, \mathcal{C}).$$

For a P system Π having a set of rules R we define *Applicable*$(\Pi, \mathcal{C})$ = *Applicable*$(R, \mathcal{C})$. Following [3] it is possible to define now the transition modes as a restriction of this set. However, it should be noted that since the corresponding multisets contain instantiated rules, additional restrictions based on instances can be placed.

Now we are ready to define the application of a multiset of rules R.

Let $\mathcal{C} = (L, \rho)$ be the current configuration and let $RI \in$ *Applicable*$(R, \mathcal{C})$, $RI = \{(r_1, v_1), \dots, (r_n, v_n)\}$ be a multiset of instantiated rules. We now define the operation *Apply*$(RI, \mathcal{C}) \in \mathfrak{C}$ which is the result of the application of RI to C.

Before giving the algorithm we remark that a rule is composed from three parts: the rewriting of objects and the label change (R), the membrane deletion (D) and the membrane creation (G). The order of the application of these parts is extremely important, e.g. doing the rewriting before the membrane creation permits to copy the result of the rewriting to the new membranes. In this article we consider that the application order is RGD, *i.e.* rewriting, creation and then deletion. This order corresponds to the actual state of art in the area of P systems with active membranes. Other orders are also possible and this can be an interesting topic for a further research.

The algorithm for the computation of *Apply*$(RI, \mathcal{C})$ is the sequence consisting of the following steps.

1. (rewriting application): $L_1 = \{(i_1, l_1, w_1') \dots (i_n, l_n, w_n')\}$ where:

$$w'_j = w_j + \bigcup_{\substack{(r_k, v_k) \in RI \\ (s, u \to v) \in Rewrite(r_k) \\ v_k|_s = j}} (-u + v).$$

2. (label change): $L_2 = \{(i_1, l'_1, w'_1) \dots (i_n, l'_n, w'_n)\}$ where:

$$l'_j = \begin{cases} e_s, \text{ there is } (r_k, v_k) \in RI \text{ such that } (s, e_s) \in \mathit{Label\text{–}Rename}(r_k) \\ \qquad \text{and } v_k|_s = j, \\ l_j, \ \text{ otherwise.} \end{cases}$$

3. (membrane creation): ($m_1 \dots m_{t+s}$ are new ids). We define the lists of newly created cells L_c and L'_c:

$$\begin{aligned} L_c(r_k) =& (m_1, h_1, u_1) \dots (m_t, h_t, u_t), \quad (r_k, v_k) \in RI \text{ and} \\ & Generate(r_k) = \{(1', h_1, u_1) \dots (t', h_t, u_t)\}. \\ L_c =& \prod_{k=1}^{n} L_c(r_k). \end{aligned}$$

$L'_c(r_k) = (m_{t+1}, h_{t+1}, w'_{n_1} - u_1 + v_1) \dots (m_{t+s}, h_{t+s}, w'_{n_s} - u_s + v_s)$, where $(r_k, v_k) \in RI$ and

$$\begin{aligned} \mathit{Generate\text{–}and\text{–}Copy}(r) =& \{((t+1)', h_{t+1}, n_{t+1}, u_{t+1} \to v_{t+1}) \dots \\ & ((t+s)', h_{t+s}, n_{t+s}, u_{t+s} \to v_{t+s})\}, \\ & (i_j, l'_j, w'_j) \in L_2, 1 \le j \le n \end{aligned}$$

$$L'_c = \prod_{k=1}^{n} L'_c(r_k).$$

By definition, we put $v_k|_{q'} = m_q$, $1 \le q \le t + s$.
We also consider a graph transducer $\mathit{CREATE\text{–}NODES}$ that creates nodes $m_1, \dots, m_{t+s}$.
We put $L_3 = L_2 \cdot L_c \cdot L'_c$ (this is the $\mathcal{C}_m$ part of the result of the application of R).

4. (membrane deletion):
Consider a vector $P = (p_1, \dots, p_n)$ defined as follows:

$$p_j = \begin{cases} *, & \text{if there exists } (r_k, v_k) \in RI \text{ such that } s \in Delete(r_k) \\ & \text{and } v_k|_s = j, \\ v_k|_m, & \text{if there exists } (r_k, v_k) \in RI \text{ such that} \\ & (s, m) \in \mathit{Delete\text{–}and\text{–}Move}(r_k) \text{ and } v_k|_s = j, \\ j, & \text{otherwise.} \end{cases}$$

The first two cases correspond to those ids j for which the corresponding cells should be deleted. We remark that for any p_k such that $p_k \neq i_k$, there is a value $z \in \mathbb{N} \cup \{*\}$ such that there is a sequence $x_1, \ldots, x_m$ with $x_1 = p_k$, $x_m = z$, and $x_j = p_{x_{j-1}}$, $2 \leq j \leq m$. We denote this by $z = last(x)$. The above affirmation follows from that fact that the *Delete–and–Move* relation (considered as a parent relation) induces a forest on the ids of the cells that should be deleted. The roots of the obtained trees are given by the function *last* and they will collect the objects from all the cells in the tree (if they are different from $*$).

Next we describe how the contents is moved:
$L_4 = \{(i_1, l'_1, w''_1) \ldots (i_n, l'_n, w''_n)\}$ where $(i_k, l'_k, w'_k) \in L_3$, $1 \leq k \leq n$, and

$$w''_j = w'_j + \bigcup_{last(k)=j} w'_k.$$

The deletion of cells induces changes to the relation ρ. We collect these modifications as a graph transducer *DELETE–NODES* that will be run after the *Change–Relation* transducer. This transducer deletes all vertices j such that $p_j \neq j$ as well as all edges that are incoming to these deleted nodes.

We also remove the corresponding cells from L_4:
$L_5 = (i_1, l'_1, w''_1) \ldots (i_{n_1}, l'_{n_1}, w''_{n_1})$ where $(i_j, l'_j, w''_j) \in L_4$ and $p_j = i_j$.

5. (relation change) The new relation $\mathcal{C}'_\rho$ is computed by running the graph transducers *CREATE–NODES*, *Change–Relation*$(r\langle v_k \rangle)$ and *DELETE–NODES* for all $(r_k, v_k) \in R$ on $\mathcal{C}_\rho$.

3 Taxonomy

In order to simplify the notation we consider several variants of rule notation:

Simple rewriting rule (R-rule)

An R-rule is defined only by the following components:
$r = (Labels(r), \rho(r), Rewrite(r))$

Simple rewriting rule with label rename (LR-rule)

An LR-rule is defined only by the following components:
$r = (Labels(r), \rho(r), Rewrite(r), Label\text{–}Rename(r))$

Simple creation rule (C-rule)

A C-rule is defined by the following components:
$r = (Labels(r), \rho(r), Generate(r), Generate\text{–}and\text{–}Copy(r),$
$\quad Change\text{–}Relation(r))$

Simple creation rule with label rename (CL-rule)

A CL-rule is defined by the following components:
$r = (Labels(r), \rho(r), Label\text{–}Rename(r), Generate(r),$
$\quad Generate\text{–}and\text{–}Copy(r), Change\text{–}Relation(r))$

Simple dissolution rule (D-rule)

A D-rule is defined by the following components:
$r = (Labels(r), \rho(r), Delete(r), Delete\text{–}and\text{–}Move(r),$
$\quad Change\text{–}Relation(r))$

In the case of the parent relation (tree case), we can simplify the rules and omit $\rho(r)$ by supposing that $Labels(r)$ is of size 2. In this case we implicitly assume that $(1, 2) \in \rho(r)$. The type of corresponding rules with parent relation will additionally contain the letter P (e.g., PC-rule).

In a more general way we can combine several components: L – label rename, R – rewriting, C – membrane creation, D – membrane deletion (and get RD rules for example).

Rules r having a non-empty $Delete\text{–}and\text{–}Move(r)$ component can be simplified by reducing their $Change\text{–}Relation(r)$ component in the case of the parent relation.

In the above case we will assume that $Change\text{–}Relation(r)$ contains the transducer $MOVE - CONNECTIONS$ described below. This transducer adds the following edges to ρ: $\{(a_x, b_y) \mid (x, y) \in \mathcal{C}_\rho \text{ and } a_x, b_y \neq *\}$, where (a_x, b_y) is defined as follows (i_m is the id of membrane m):

$$(a_x, b_y) = \begin{cases} (last(x), y), \ (x, y) \in \rho \text{ and } p_x \neq i_x, \\ (y, last(x)), \ (y, x) \in \rho \text{ and } p_x \neq i_x. \end{cases}$$

The above transformations correspond to the deletion of cells and to the movement of their contents according to $Delete\text{–}and\text{–}Move$ relation.

4 Some examples

4.1 Active membranes

Let us start with the example of traditional active membrane rules (e.g., see Section 11.2 from handbook).

Polarization can be treated in two ways – as a special object inside a membrane or like a special label; we here consider the latter case, i.e., the couple (label,polarization) will be a new type of label.

Thus, a rule $r : [a \to v]_h^e$ will be treated as $r : [a \to v]_{\langle e,h \rangle}$ and it can be translated as the following PR-rule:

$$\begin{aligned} r: \qquad \mathit{Labels}(r) &= (\langle e,h\rangle), \\ \mathit{Rewrite}(r) &= \{(1, a \to v)\}. \end{aligned}$$

In the future we indicate e instead of $\langle e,h\rangle$.

A rule $a[]_{e_1} \to [b]_{e_2}$ can be translated as the following group of PLR-rules ($\forall p \in \mathit{Lab}$):

$$\begin{aligned} r: \qquad \mathit{Labels}(r) &= (e_1, p), \\ \mathit{Rewrite}(r) &= (2, a) \to (1, b), \\ \mathit{Label\text{–}Rename}(r) &= \{(1, e_2)\}. \end{aligned}$$

A rule $[a]_{e_1} \to []_{e_2} b$ can be translated as the following group of PLR-rules ($\forall p \in \mathit{Lab}$):

$$\begin{aligned} r: \qquad \mathit{Labels}(r) &= (e_1, p), \\ \mathit{Rewrite}(r) &= (1, a) \to (2, b), \\ \mathit{Label\text{–}Rename}(r) &= \{(1, e_2)\}. \end{aligned}$$

A rule $[a]_e \to b$ can be translated as the following group of PD-rules ($\forall p \in \mathit{Lab}$):

$$\begin{aligned} r: \qquad \mathit{Labels}(r) &= (e, p), \\ \mathit{Rewrite}(r) &= \{(1, a) \to (1, b)\}, \\ \mathit{Delete\text{–}and\text{–}Move}(r) &= \{(1, 2)\}. \end{aligned}$$

A rule $[a]_{e_1} \to [b]_{e_2}[c]_{e_3}$ can be translated as the following group of $PCLR$-rules ($\forall p \in \mathit{Lab}$):

$$\begin{aligned} r: \qquad \mathit{Labels}(r) &= (e, p), \\ \mathit{Rewrite}(r) &= (1, a) \to (1, b), \\ \mathit{Label\text{–}Rename}(r) &= \{(1, e_2)\}, \\ \mathit{Generate\text{–}and\text{–}Copy}(r) &= \{(1', e_3, 1, b \to c)\}, \\ \mathit{Change\text{–}Relation}(r) &= INSERT - EDGE(1', 2). \end{aligned}$$

4.2 Rules without polarizations

(According to Section 11.4 from the handbook). Since in our case the label is a couple $\langle e,h\rangle$, there is no distinction with respect to the previous case.

4.3 Creation rules

Consider creation rules like on p. 326 in handbook.

A rule $[a \to [u]_{h_1}]_{h_2}$ can be translated as following PCR-rule:

$$\begin{aligned} r: \qquad \mathit{Labels}(r) &= (h_2), \\ \mathit{Rewrite}(r) &= (1, a) \to (1, \lambda), \\ \mathit{Generate}(r) &= \{(1', h_1, u)\}, \\ \mathit{Change\text{–}Relation}(r) &= INSERT - EDGE(1', 1). \end{aligned}$$

4.4 Strong division

A rule $[[]_{h_1}\dots[]_{h_k}[]_{h_{k+1}}\dots[]_{h_n}]_h \rightarrow [[]_{h_1}\dots[]_{h_k}]_h[[]_{h_{k+1}}\dots[]_{h_n}]_h$ can be defined as the following C-rule:

$$
\begin{aligned}
r: \qquad \mathit{Labels}(r) &= (h_1,\dots,h_n,h),\\
\rho(r) &= \{(i,n+1) \mid 1 \le i \le n\},\\
\mathit{Rewrite}(r) &= \emptyset\\
\mathit{Generate}(r) &= \{(1',h,\lambda)\},\\
\mathit{Generate\text{–}and\text{–}Copy}(r) &= \emptyset,\\
\mathit{Change\text{–}Relation}(r) &= \mathit{DELETE}-\mathit{EDGE}(k,n+1), i+1 \le k \le n, \text{ and}\\
&\quad\ \mathit{INSERT}-\mathit{EDGE}(k,1').
\end{aligned}
$$

4.5 Division based on polarizations

Consider a rule of type $[]_h \rightarrow [+]_h[-]_{h2}[0]_{h3}$ that regroups all membranes with the same polarization in three new membranes. This can be simulated with the following C-rule:

$$
\begin{aligned}
r: \qquad \mathit{Labels}(r) &= (h),\\
\rho(r) &= \emptyset,\\
\mathit{Rewrite}(r) &= \emptyset,\\
\mathit{Generate}(r) &= \{(1',h_1,\lambda),(2',h_2,\lambda)\},\\
\mathit{Generate\text{–}and\text{–}Copy}(r) &= \emptyset,\\
\mathit{Change\text{–}Relation}(r) &= \begin{array}{l}
\mathit{DELETE\text{–}EDGE}(k,1), \text{ and } \mathit{INSERT\text{–}EDGE}(k,1'),\\
\text{ for all } k \text{ such that } \mathit{lab}(k) = -\\
\mathit{DELETE\text{–}EDGE}(k,1), \text{ and } \mathit{INSERT\text{–}EDGE}(k,2'),\\
\text{ for all } k \text{ such that } \mathit{lab}(k) = 0
\end{array}
\end{aligned}
$$

5 Conclusions

In this paper we presented a framework for P systems with dynamic structure. The obtained meta-language has a precise semantics centered around 2 notions: (1) the evolution of the objects and membrane labels and (2) the evolution of the membrane structure (creation and deletion of nodes and edges). As a consequence it permits to easily describe different features of existing P systems with dynamical structure, which permits to provide an interesting tool for the comparison of different variants of P systems. Moreover, the translation to the framework allows for a better understanding of the corresponding P system and provides ways to extend its definition by new features. We remark that in the case of the systems with a static structure a similar approach using the framework from [3] permitted to define new variants of P systems and to better express some existing ones [1, 4].

The introduced model works with an arbitrary (binary) relation between membranes, so it could be interesting to consider relations different from the parent relation widely used in P systems. As an interesting candidate we suggest the brother/sister relation on a tree. It could also be interesting to consider a generalization of the framework to an arbitrary n-ary relation. In this case the relation ρ induces a hypergraph, so the components changing the structure of ρ have to be adapted to work on hypergraphs.

Another direction for the development of the framework is to consider that for a multiset of rules the order of the application of *Change–Relation* produces different results. This implies that the order of rules is important, by consequence the set $Applicable(\Pi, C, \delta)$ will contain vectors (or lists) of rules. This interesting idea was not yet considered in the framework of P systems and we think that it can lead to interesting results.

References

1. A. Alhazov, M. Oswald, R. Freund, S. Verlan, Partial Halting and Minimal Parallelism Based on Arbitrary Rule Partitions, *Fundamenta Informaticae* **91**(1), 2009, 17–34.
2. R. Freund, B. Haberstroh, Attributed Elementary Programmed Graph Grammars, Proceedings 17th Intern. Workshop on Graph-Theoretic Concepts in Computer Science, *Lecture Notes in Computer Science* **570**, Springer, 1991, 75–84.
3. R. Freund, S. Verlan, A Formal Framework for Static (Tissue) P Systems, Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **4860** , 271–284, Springer, 2007.
4. R. Freund, S. Verlan, (Tissue) P systems working in the k-restricted minimally or maximally parallel transition mode, *Natural Computing* **10**(2), 2011, 821–833.
5. Gh. Păun, *Membrane Computing. An Introduction.* Springer–Verlag, 2002.
6. G. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook Of Membrane Computing.* Oxford University Press, 2009.
7. *The Membrane Computing Web Page*: `http://ppage.psystems.eu`
8. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages.* Springer–Verlag, Berlin, 1997.

P Systems with Minimal Left and Right Insertion and Deletion

Rudolf Freund[1], Yurii Rogozhin[2], and Sergey Verlan[3]

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

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

[3] LACL, Département Informatique, Université Paris Est
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@univ-paris12.fr

Summary. In this article we investigate the operations of insertion and deletion performed at the ends of a string. We show that using these operations in a P systems framework (which corresponds to using specific variants of graph control), computational completeness can even be achieved with the operations of left and right insertion and deletion of only one symbol.

1 Introduction

The operations of left and right insertion and deletion that we consider in this article correspond to the operations of left and right concatenation and quotient with a finite language. While these operations are known for a long time, their joint investigation in a distributed framework originates from the area of natural computing, where they were used in the context of networks of evolutionary processors (NEP) [7]. Such networks are a special type of networks of language processors [6] that feature a set of (rewriting) nodes rewriting languages and after that redistributing some regular subsets between the nodes. In networks of evolutionary processors, the rewriting operations are replaced by three types of operations having a biological motivation: insertion, deletion, and mutation (substitution). The corresponding systems are quite powerful and we refer to [8] for more details. The redistribution of the node contents based on a regular condition is a very powerful operation. Accepting hybrid networks of evolutionary processors (AHNEP) replace this condition by random context conditions, however, the set of operations is changed and now includes the insertion and deletion operations at the extremities of the strings; we refer to [21, 9] for more details on AHNEP.

The operations of insertion and deletion on the extremities of a string can also be seen as a particular case of a more general variant, where insertion and deletion can be performed anywhere in the string. The insertion operation defined in such a way was first considered in [14, 15] and after that related insertion and deletion operations were investigated in [17, 18]. Another generalization of the insertion and deletion operations that involves the checking of contexts for the insertion and deletion was considered with a linguistic motivation in [13, 20] and with a biological motivation in [4, 5, 18, 26]. Generally, if the length of the contexts and/or of the inserted and deleted strings are big enough, then the insertion-deletion closure of a finite language leads to computational completeness. There are numerous results establishing the descriptional complexity parameters sufficient to achieve this goal, we refer to [31, 30] for an overview of this area.

Some descriptional complexity parameters lead to variants that are not computationally complete. An investigation of insertion and deletion operations combined with regulating mechanisms was done for these cases, more precisely, with the graph-controlled, the matrix, and the random-context controls [11, 28, 16]. As it was shown in these articles, in most of the cases the additional control leads to computational completeness. The graph-controlled regulation is of particular interest, as it can be related to the notion of P systems. Such systems formalize the functioning of a living cell that topologically delimits processing units by membranes, thus leading to a tree (or graph) structure of processing nodes. The elements processed in some node (membrane) then are distributed among the neighbors in the structure. We refer to [24, 25] and to the web page [29] for more details on P systems. In the case of the operations of insertion and deletion acting on strings this directly corresponds to a graph control where the control nodes correspond to the membranes.

The research on context-free insertion and deletion (i.e., without contextual dependency) shows that if the lengths of the inserted and deleted strings are 2 and 3 (or 3 and 2), respectively, then the insertion-deletion closure of finite languages is computationally complete [22]. When one of these parameters is decreased, this result is not true anymore [32]; moreover, even the graph-controlled variant cannot achieve computational completeness [19]. This changes when a graph control with appearance checking is used [1] or in the case of a random context control [16]. In both variants, minimal operations (involving only one symbol) were considered, leading to RE (the family of recursivey enumerable languages) in the case of set-controlled random context conditions and to $PsRE$ (the family of Parikh sets of RE) in the case of graph control with appearance checking.

We note that the operations of left and right insertion and deletion are incomparable with normal insertion and deletion: because of the positional information, the regular language a^+b^+ can be obtained even with left and right insertions of only one symbol, yet not when insertions are possible at arbitrary positions in the string. On the other hand, the Dyck language cannot be obtained when insertion is only possible at the ends of the strings, while with normal insertion this can be done easily. In [1, 3], left and right insertion and deletion operations (under the name of exo-insertion and -deletion) were considered in the P systems framework (i.e., with a graph control) and it was shown that systems with insertion of strings of length 2 (respectively 1) and deletion of strings of length 1 (respectively 2) lead to computational completeness. In the case of minimal insertion and deletion (i.e., of only one symbol), a priority of deletion over insertion (corresponding to an appearance check) was used to show computational completeness.

In this article we continue these investigations and we consider P systems with minimal left and right insertion and deletion and prove that computational completeness can be achieved even in this case, with the structure of the P system we need being matrix-like. We also directly show that matrix grammars using minimal left insertion and minimal right deletion rules are computationally complete (with matrices of length at most 3). Moreover, we also prove that using an additional minimal mutation operation (substitution of one symbol by another one) allows for reducing the height of the tree structure of the P system to the minimum size 1.

2 Preliminaries

After some preliminaries from formal language theory, we define the string rewriting rules to be used in this paper. As string rewriting systems, we will consider Post systems, matrix grammars, and sequential P systems. Moreover, we will give some examples and preliminary results to illustrate our definitions.

The set of non-negative integers is denoted by $\mathbb{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the elements of V^* are called strings, and the *empty string* is denoted by λ; $V^* \setminus \{\lambda\}$ is denoted by V^+. Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the number of occurrences of all symbols from V in x is denoted by $|x|$. The family of recursively enumerable string languages is denoted by RE. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as [10, 27].

We here consider string rewriting rules only working at the ends of a string:

Post rewriting rule $P[x/y]$ with $x, y \in V^*$: $P[x/y](wx) = yw$ for $w \in V^*$.
Left substitution $S_L[x/y]$ with $x, y \in V^*$: $S_L[x/y](xw) = yw$ for $w \in V^*$.
Right substitution $S_R[x/y]$ with $x, y \in V^*$: $S_R[x/y](wx) = wy$ for $w \in V^*$.

If in a (left or right) substitution $S_L[x/y]$ or $S_R[x/y]$ x is empty, then we call it an *insertion* and write $I_L[y]$ and $I_R[y]$, respectively; if in a (left or right) substitution $S_L[x/y]$ or $S_R[x/y]$ y is empty, then we call it a *deletion* and write $D_L[x]$ and $D_R[x]$, respectively. If we only insert one symbol a, then we will also write $+a$, $a+$, $-a$, and $a-$ for $I_L[a]$, $I_R[a]$, $D_L[a]$, and $D_R[a]$, respectively.

In general, a *(string rewriting) grammar* G of type X is a construct (V,T,A,P) where V is a set of *symbols*, $T \subseteq V$ is a set of *terminal symbols*, $A \in V^+$ is the *axiom*, and P is a finite set of *rules* of type X. Each rule $p \in P$ induces a relation $\Longrightarrow_p \subseteq V^* \times V^*$; p is called *applicable* to a string $x \in V^*$ if and only if there exists at least one string $y \in V^*$ such that $(x,y) \in \Longrightarrow_p$; we also write $x \Longrightarrow_p y$. The derivation relation $\Longrightarrow_G$ is the union of all $\Longrightarrow_p$, i.e., $\Longrightarrow_G = \cup_{p\in P} \Longrightarrow_p$. The reflexive and transitive closure of $\Longrightarrow_G$ is denoted by $\overset{*}{\Longrightarrow}_G$.

The *language generated by* G is the set of all terminal strings derivable from the axiom, i.e., $L(G) = \left\{ v \in T^* \mid A \overset{*}{\Longrightarrow}_G v \right\}$. The family of languages generated by grammars of type X is denoted by $\mathcal{L}(X)$.

In general, we write $S_R^{k,m}$ for a type of grammars using only substitution rules $S_R[x/y]$ with $|x| \leq k$ and $|y| \leq m$. In the same way, we define the type $S_L^{k,m}$ for a type of grammars using only substitution rules $S_L[x/y]$ with $|x| \leq k$ and $|y| \leq m$, as well as the types I_L^m, I_R^m, D_L^k, and D_R^k, respectively. The type $D^k I^m$ allows for the deletion of strings with length $\leq k$ and for the insertion of strings with length $\leq m$. If, in addition, we also allow substitutions $S_R[x/y]$ with $|x| \leq k'$ and $|y| \leq m'$, we get the type $D^k I^m S^{k'm'}$; we observe that the type $D^k I^m S^{k'm'}$ is subsumed by the type $S^{k'm'}$ if $k \leq k'$ and $m \leq m'$. If we allow the parameters k and/or m to be arbitrarily large, we just omit them, e.g., DI is the type allowing to use deletions and insertions of strings of arbitrary lengths.

Example 1. Let $G = (V,T,A,P)$ be a regular grammar, i.e., the rules in P are of the form $A \to bC$ and $A \to \lambda$ with $A,C \in V \setminus T$ and $b \in T$. Then the grammar $G' = (V,T,A,\{S_R[A/y] \mid A \to y \in P\})$ with substitution rules generates the same language as G, i.e., $L(G') = L(G)$. Hence, with REG denoting the family of regular languages, we obviously have got $REG \subseteq \mathcal{L}\left(S_R^{1,2}\right)$. □

It is not difficult to check that grammars of type $D^1 I^1 S^1$ have a rather limited computational power. Indeed, we can show the following representation of languages generated by grammars of type $D^1 I^1 S^1$:

Theorem 1. *Every language $L \subseteq T^*$ in $\mathcal{L}\left(D^1 I^1 S^1\right)$ can be written in the form $T_l^* S T_r^*$ where $T_l, T_r \subseteq T$ and S is a finite subset of T^*.*

Proof. Let $G = (V,T,A,P)$ be a grammar of type $D^1 I^1 S^1$ and let $N := V \setminus T$. We first construct the start set S as follows: Consider all possible derivations in G from A with only using substitutions and deletions, but without loops, i.e., no string is allowed to appear more than once in such a derivation, which means that all these derivations are of bounded length (bounded by the number of strings in V of length at most $|V|$).Then S consists of all terminal strings obtained in this way (finding these strings is a finitely bounded process, as to each of the possible strings in V of length at most $|V|$, at most $|P|$ rules can be applied). A symbol from N remaining inside a string blocks that string from ever becoming terminal by applying rules from P, and deletion of a symbol can be avoided by just not introducing the symbol which by a sequence of minimal substitutions would lead to the symbol to be deleted. Hence, for constructing the sets T_l (T_r, respectively) we can restrict ourselves to the terminal symbols b either directly inserted by minimal insertion rules $I_l\,[b]$ ($I_r\,[b]$, respectively) or obtained by a sequence of one minimal insertion together with a bounded (by $|V|$) number of minimal substitutions $S_l\,[a/b]$ ($S_r\,[a/b]$, respectively).

Therefore, in sum $L\,(G)$ can be written as the finite union of languages generated by grammars of type I^1, i.e., $L\,(G) = \cup_{w\in S} L\,(G_w)$ where

$$G_w = (T, T, w, \{I_l\,[b] \mid b \in T_l\} \cup \{I_r\,[b] \mid b \in T_r\}).$$

In fact, this representation of languages in $\mathcal{L}\left(D^1I^1S^1\right)$ means that for the type $D^1I^1S^1$ we could forget minimal deletions and substitutions and instead consider finite subsets of axioms instead of a single axiom. Putting an A in front of the types for this variant of grammars, we just have proved that

$$\mathcal{L}\left(A\text{-}D^1I^1S^1\right) = \mathcal{L}\left(A\text{-}I^1\right). \qquad \square$$

2.1 Post Systems

A *Post system* is a grammar using only Post rewriting rules (a grammar of *type PS*). A Post system (V, T, A, P) is said to be in *normal form* (a grammar of *type PSNF*) if and only if the Post rewriting rules $P\,[x/y]$ in P are only of the forms $P\,[ab/c]$, $P\,[a/bc]$, $P\,[a/b]$, and $P\,[a/\lambda]$, with $a, b, c \in V$. A Post system (V, T, A, P) is said to be in *Z-normal form* (a grammar of *type PSZNF*) if and only if it is in normal form and, moreover, there exists a special symbol $Z \in V \setminus T$ such that

- Z appears only once in the string x of a Post rewriting rule $P\,[x/y]$, and this rule is $P\,[Z/\lambda]$;
- if the rule $P\,[Z/\lambda]$ is applied, the derivation in the Post system stops yielding a terminal string;
- a terminal string can only be obtained by applying the rule $P\,[Z/\lambda]$.

Although basic results concerning Post systems are folklore since many years, e.g., see [23], we need the special Z-normal form for the proof of our main theorem; the following result is an immediate consequence of the proof given in [12] for Lemma 1 there:

Theorem 2. *For every recursively enumerable language $L \subseteq T^*$ there exists a Post rewriting system G, $G = (V, T, A, P)$, in Z-normal form such that $L\,(G) = L$, i.e., $\mathcal{L}\,(PS) = \mathcal{L}\,(PSNF) = \mathcal{L}\,(PSZNF) = RE$.*

2.2 Matrix Grammars

A *matrix grammar* of type X is a construct $G_M = (G, M)$ where $G = (V, T, A, P)$ is a grammar of type X, M is a finite set of sequences of the form $(p_1, \dots, p_n)$, $n \geq 1$, of rules in P. For $w, z \in V^*$ we write $w \Longrightarrow_{G_M} z$ if there are a matrix

$(p_1, \ldots, p_n)$ in M and objects $w_i \in V^*$, $1 \le i \le n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \le i \le n$, $w_i \Longrightarrow_G w_{i+1}$. The maximal length n of a matrix $(p_1, \ldots, p_n) \in M$ is called the degree of G_M.

$L(G_M) = \left\{ v \in T^* \mid A \Longrightarrow_{G_M}^* v \right\}$ is the language generated by G_M. The family of languages generated by matrix grammars of type X (of degree at most n) is denoted by $\mathcal{L}(X\text{-}MAT)$ $(\mathcal{L}(X\text{-}MAT_n))$.

Theorem 3. *$\mathcal{L}\left(D^2I^2\text{-}MAT_2\right) = \mathcal{L}\left(D^1I^1\text{-}MAT_3\right) = \mathcal{L}(PSNF) = RE$.*

Proof. From Theorem 2 we know that $\mathcal{L}(PSNF) = RE$, hence, we will only show that for every Post system $G = (V, T, A, P)$ in normal form we are able to construct equivalent matrix grammars $G_1 = (G, M_1)$ and $G_2 = (G, M_2)$ of type D^2I^2 and of type D^1I^1, respectively:

$$\begin{aligned}
M_1 &= \{(D_R[x], I_L[y]) \mid P[x/y] \in P\}, \\
M_2 &= \{(D_R[b], D_R[a], I_L[c]) \mid P[ab/c] \in P\} \\
&\cup \{(D_R[a], I_L[c], I_L[b]) \mid P[a/bc] \in P\} \\
&\cup \{(D_R[a], I_L[b]) \mid P[a/b] \in P\} \\
&\cup \{(D_R[a]) \mid P[a/\lambda] \in P\}.
\end{aligned}$$

As each rule in G is directly simulated by a matrix in M_1 and in M_2, respectively, we immediately infer $L(G) = L(G_1) = L(G_2)$. □

Whereas the matrices in M_1 are only of length 2, the degree of M_2 is 3; it remains as an open question whether also with rules of type D^1I^1 we could decrease the degree to 2 or not; we conjecture that the answer is *no*. As we have shown in Theorem 1, with grammars using rules of type $D^1I^1S^1$ we are not able to obtain RE, we even remain below the regular language class; hence, we need such regulating mechanisms as matrices to reach computational compleness.

2.3 P Systems

We now introduce another variant to guide the derivations in a grammar using rules of those types introduced above, i.e., specific variants of left and right substitution rules.

A *(sequential) P system of type X with tree height n* is a construct $\Pi = (G, \mu, R, i_0)$ where $G = (V, T, A, P)$ is a grammar with rules of type X and

- μ is the membrane (tree) structure of the system with the height of the tree being n (μ usually is represented by a string containing correctly nested marked parentheses); we assume the membranes, i.e., the nodes of the tree representing μ, being uniquely labelled by labels from a set Lab;
- R is a set of rules of the form (h, r, tar) where $h \in Lab$, $r \in P$, and tar, called the *target indicator*, is taken from the set $\{here, in, out\} \cup \{in_j \mid 1 \le j \le n\}$; the rules assigned to membrane h form the set $R_h = \{(r, tar) \mid (h, r, tar) \in R\}$, i.e., R can also be represented by the vector $(R_h)_{h \in Lab}$;

- i_0 is the initial membrane where the axiom A is put at the beginning of a computation.

As we only have to follow the trace of a single string during a computation of the P system, a configuration of $\mathit{\Pi}$ can be described by a pair (w, h) where w is the current string and h is the label of the membrane currently containing the string w. For two configurations (w_1, h_1) and (w_2, h_2) of $\mathit{\Pi}$ we write $(w_1, h_1) \Longrightarrow_{\mathit{\Pi}} (w_2, h_2)$ if we can pass from (w_1, h_1) to (w_2, h_2) by applying a rule $(h_1, r, \mathit{tar}) \in R$, i.e., $w_1 \Longrightarrow_r w_2$ and w_2 is sent from membrane h_1 to membrane h_2 according to the target indicator tar. More specifically, if $\mathit{tar} = \mathit{here}$, then $h_2 = h_1$; if $\mathit{tar} = \mathit{out}$, then the string w_2 is sent to the region h_2 immediately outside membrane h_1; if $\mathit{tar} = \mathit{in}_{h_2}$, then the string is moved from region h_1 to the region h_2 immediately inside region h_1; if $\mathit{tar} = \mathit{in}$, then the string w_2 is sent to one of the regions immediately inside region h_1.

A sequence of transitions between configurations of $\mathit{\Pi}$, starting from the initial configuration (A, i_0), is called a *computation* of $\mathit{\Pi}$. A *halting computation* is a computation ending with a configuration (w, h) such that no rule from R_h can be applied to w anymore; (w, h) is called the result of this halting computation if $w \in T^*$. $L(\mathit{\Pi})$, the language generated by $\mathit{\Pi}$, consists of all strings over T which are results of a halting computation in $\mathit{\Pi}$.

By $\mathcal{L}(X\text{-}LP)$ $(\mathcal{L}(X\text{-}LP^{\langle n\rangle}))$ we denote the family of languages generated by P systems (of tree height at most n) using rules of type X. If only the targets *here*, *in*, *out* are used, then the P system is called *simple*, and the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}LsP)$ $(\mathcal{L}(X\text{-}LsP^{\langle n\rangle}))$.

Example 2. Let $\mathit{\Pi} = (G, [_1\, [_2\,]_2\, [_3\,]_3\, [_4\,]_4\,]_1, R, 1)$ be a P system of type $D_R^1 I_L^2$ with

$$\begin{aligned} G &= (\{a, B\}, \{a\}, \{D_R[a], D_R[B], I_L[aa], I_L[B]\}, aB), \\ R &= \{(1, D_R[a], \mathit{in}_2), (1, D_R[B], \mathit{in}_3), (1, D_R[B], \mathit{in}_4)\} \\ &\quad \cup \{(2, I_L[aa], \mathit{out}), (3, I_L[B], \mathit{out})\} \end{aligned}$$

The computations in $\mathit{\Pi}$ start with aB in membrane (region) 1. In general, starting with a string $a^{2^n}B$, $n \geq 0$, in membrane 1, we may either delete B by the rule $(1, D_R[B], \mathit{in}_4)$, getting a^{2^n} as the terminal result in the elementary membrane 4 (a membrane is called *elementary* if and only if it contains no inner membrane) or delete B by the rule $(1, D_R[B], \mathit{in}_3)$. With the string a^{2^n} arriving in membrane 3, we get Ba^{2^n} in membrane 1 by the rule $(3, I_L[B], \mathit{out})$. Now we double the number of symbols a by applying the sequence of rules $(1, D_R[a], \mathit{in}_2)$ and $(3, I_L[aa], \mathit{out})$ 2^n times, finally obtaining $a^{2^{n+1}}B$. Hence, in sum we get $L(\mathit{\Pi}) = \{a^{2^n} \mid n \geq 0\}$ for the language generated by this P system μ of type $D_R^1 I_L^2$. □

3 Computational Completeness of P Systems with Minmal Substitution Rules

In this section we consider several variants of P systems with substitution rules of minimal size, the main result showing computational completeness for simple P systems with rules of type D^1I^1. Yet first we show that for any recursively enumerable language we can construct a P system, with the height of the tree structure being only 1 (which is the minimum possible according to Theorem 1), of type $D^1_R I^1_L S^1_R$, i.e., using minimal right insertions and minimal right deletions and mutations (substitutions).

Theorem 4. *$\mathcal{L}\left(D^1_R I^1_L S^1_R\text{-}LP^{\langle 1\rangle}\right) = RE$.*

Proof. From Theorem 2 we know that $\mathcal{L}(PSZNF) = RE$, hence, we will only show that for every Post system $G = (V, T, A, P)$ in Z-normal form we are able to construct equivalent P system $\mathit{\Pi}$ of type $D^1_R I^1_L S^1_R$. We assume that the rules in P are labelled in a unique way by labels from a finite set Lab with $1 \notin Lab$ and $z \in Lab$. We now construct a P system $\mathit{\Pi}$, $\mathit{\Pi} = (G', \mu, R, 1)$, with a flat tree structure μ of height 1, i.e., with the outermost membrane (the so-called skin membrane) being labelled by 1, and all the other membranes being elementary membranes inside the skin membrane being labelled by labels from

$$\begin{aligned} Lab' = {} & \{1, \#\} \cup \{l \mid l : p \in Lab\} \\ & \cup \left\{\bar{h} \mid h : P\left[a_h/b_h c_h\right] \in P\right\} \cup \left\{\bar{h} \mid h : P\left[a_h b_h/c_h\right] \in P\right\}. \end{aligned}$$

$G' = (V', T, A, P')$, $V' = \left\{x, \bar{x}^l \mid x \in V, l \in Lab\right\} \cup \{\#\}$, and P' contains the minimal left insertion, right deletion, and right substitution rules contained in the rules of R as listed in the following:

$$\begin{array}{ll}
h : P\left[a_h b_h/c_h\right]: & \left(1, D_R\left[b_h\right], in_h\right),\ \left(h, S_R\left[a_h/\bar{a}^h_h\right], out\right),\ \left(h, I_L\left[\#\right], out\right), \\
 & \left(1, D_R\left[\bar{a}^h_h\right], in_{\bar{h}}\right),\ \left(\bar{h}, I_L\left[c_h\right], out\right); \\
h : P\left[a_h/b_h c_h\right]: & \left(1, S_R\left[a_h/\bar{a}^h_h\right], in_h\right),\ \left(h, I_L\left[c_h\right], out\right), \\
 & \left(1, D_R\left[\bar{a}^h_h\right], in_{\bar{h}}\right),\ \left(\bar{h}, I_L\left[b_h\right], out\right); \\
h : P\left[a_h/b_h\right]: & \left(1, D_R\left[a_h\right], in_h\right),\ \left(h, I_L\left[b_h\right], out\right); \\
h : P\left[a_h/\lambda\right]: & \left(1, S_R\left[a_h/a_h\right], in_h\right),\ \left(l, D_R\left[a_h\right], out\right), \text{ for } a_h \neq Z; \\
z : P\left[Z/\lambda\right]: & \left(D_R\left[Z\right], in_z\right);
\end{array}$$

the additional membrane $\#$ is used to trap all computations not leading to a terminal string in an infinite loop by the rules $\left(1, I_L\left[\#\right], in_{\#}\right)$ and $\left(\#, I_L\left[\#\right], out\right)$; for this purpose, the rule $\left(h, I_L\left[\#\right], out\right)$ is used in case of $h : P\left[a_h b_h/c_h\right]$, too. Due to the features of the underlying Post system in Z-normal form, all terminal strings from $L(G)$ can be obtained as final results of a halting computation in the elementary membrane z, whereas all other possible computations in $\mathit{\Pi}$ never halt, finally being trapped in an infinite loop guaranteed by the rules leading into and out from membrane $\#$. Hence, in sum we get $L(\mathit{\Pi}) = L(G)$. □

Summarizing the results of Theorems 1 and 4, we get:

Corollary 1. $\mathcal{L}\left(D^1I^1S^1\right) = \mathcal{L}\left(D^1I^1S^1\text{-}LP^{\langle 0\rangle}\right) \subset REG \subset$
$\mathcal{L}\left(D^1_R I^1_L S^1_R\text{-}LP^{\langle n\rangle}\right) = RE$ *for all* $n \geq 1$.

If we want to restrict ourselves to the simple targets *here*, *in*, *out*, then we have to use a more difficult proof technique than in the proof of Theorem 4.

Theorem 5. $\mathcal{L}\left(D^1I^1\text{-}LsP^{\langle 8\rangle}\right) = RE$.

Proof. In order to show the inclusion $RE \subseteq \mathcal{L}\left(D^1I^1\text{-}LsPLsP^{\langle 8\rangle}\right)$, as in the proof of Theorem 4 we start from a Post system $G = (V, T, A, P)$ in Z-normal form with assuming the rules in P to be labelled in a unique way by labels from a finite set Lab with $1 \notin Lab$ and $z \in Lab$ and construct an equivalent simple P system Π, $\Pi = (G', \mu, R, 1)$, of type D^1I^1, with $G' = (V', T, A, P')$ and

$V' = V \cup V_R \cup \{S\}$, $V_R = \{D, E, F, H, J, K, M\}$,
$P' = \{+X, -X \mid X \in V \cup \{S\}\} \cup \{X+, X- \mid X \in V \cup V_R\}$,

as follows: The membrane structure μ consists of the skin membrane 1 as well as of linear structures needed for the simulation of the rules in G: For every rule $h : P\left[a_h b_h / c_h\right]$ and every rule $h : P\left[a_h / b_h c_h\right]$ in P we need a linear structure of 8 membranes $\left[_{(h,1)} \left[_{(h,2)} \cdots \left[_{(h,8)} \ \right]_{(h,8)} \cdots \right]_{(h,2)} \right]_{(h,1)}$ and for every rule $h : P\left[a_h / b_h\right]$ and every rule $h : P\left[a_h / \lambda\right]$ in P we need a linear structure of 6 membranes $\left[_{(h,1)} \left[_{(h,2)} \cdots \left[_{(h,6)} \ \right]_{(h,6)} \cdots \right]_{(h,2)} \right]_{(h,1)}$; moreover, for getting the terminal results, we need the linear structure of 3 membranes $\left[_{(z,1)} \left[_{(z,2)} \left[_{(z,3)} \ \right]_{(z,3)} \right]_{(z,2)} \right]_{(z,1)}$.

The simulations of the other rules from P are accomplished by the procedures as shown in the tables below, where the columns have to be interpreted as follows: in the first column, the membrane (label) h is listed, in the second one only the rule $p \in P$ is given, which in total describes the rule $(h, p, in) \in R$, whereas the rule p in the fifth column has to be interpreted as the rule $(h, p, out) \in R$.; the strings in the third and the fourth column list the strings obtained when going up in the linear membrane structure with the rules (h, p, in) from column 2 and going down with the rules (h, p, out) from column 5, respectively. The symbol F cannot be erased anymore, hence, whenever F has been introduced, at some moment, the computation will land in an infinite loop with only introducing more and more symbols F. The main idea of the proof is that we choose the membrane to go into by the rule $(1, K+, in)$ in a non-deterministic way. The goal is to reach the terminal membrane $(z, 3)$ starting with a string wZ, $w \in T^*$, in the skin membrane:

$(z,3)$		w		
$(z,2)$	$Z-$	wZ		$F+$
$(z,1)$	$K-$	wZK	wF	$F+$
1	$K+$	wZ	wFF	

Getting the terminal string $w \in T^*$

The tables below are to be interpreted in the same way as above; yet now we only list the results of correct simulations in column 4 and omit the results of adding the trap symbol F. Moreover, the rule $D-$ in the skin membrane is the only one in the whole system which uses the target *here*, i.e., it has to be interpreted as $(1, D-, here)$.

$(h,8)$		$ScwDH$		$H-$, $F+$
$(h,7)$	$+S$	$cwDH$	$ScwD$	$E+$, $F+$
$(h,6)$	$+c$	wDH	$ScwDE$	$M+$, $F+$
$(h,5)$	$H+$	wD	$ScwDEM$	$-S$, $F+$
$(h,4)$	$D+$	w	$cwDEM$	$M-$, $F+$
$(h,3)$	$a-$	wa	$cwDE$	$J+$, $F+$
$(h,2)$	$b-$	wab	$cwDEJ$	$J-$, $F+$
$(h,1)$	$K-$	$wabK$	$cwDE$	$E-$, $F+$
1	$K+$	wab	cwD cw	$D-$

Simulation of $h : P\,[ab/c]$

$(h,8)$		$SbcwDH$		$H-$, $F+$
$(h,7)$	$+S$	$bcwDH$	$SbcwD$	$E+$, $F+$
$(h,6)$	$+b$	$cwDH$	$SbcwDE$	$M+$, $F+$
$(h,5)$	$+c$	wDH	$SbcwDEM$	$-S$, $F+$
$(h,4)$	$H+$	wD	$bcwDEM$	$M-$, $F+$
$(h,3)$	$D+$	w	$bcwDE$	$J+$, $F+$
$(h,2)$	$a-$	wa	$bcwDEJ$	$J-$, $F+$
$(h,1)$	$K-$	waK	$bcwDE$	$E-$, $F+$
1	$K+$	wa	$bcwD$	$D-$
			bcw	

Simulation of $h : P\,[ab/c]$

$(r,6)$		$SwDH$		$H-$, $F+$
$(r,5)$	$+S$	wDH	SwD	$E+$, $F+$
$(r,4)$	$H+$	wD	$SwDE$	$S-$, $F+$
$(r,3)$	$D+$	w	wDE	$J+$, $F+$
$(r,2)$	$a-$	wa	$wDEJ$	$J-$, $F+$
$(r,1)$	$K-$	waK	wDE	$E-$, $F+$
1	$K+$	wa	wD w	$D-$

Simulation of $h : P\,[a/\lambda]$, $a \neq Z$

$(h,6)$		$SbwD$		$D-$, $F+$
$(h,5)$	$+S$	wD	Sbw	$E+$, $F+$
$(h,4)$	$+b$	wD	$SbwE$	$-S$, $F+$
$(h,3)$	$D+$	w	bwE	$J+$, $F+$
$(h,2)$	$a-$	wa	$bwEJ$	$J-$, $F+$
$(h,1)$	$K-$	waK	bwE	$E-$, $F+$
1	$K+$	wa	bw	

Simulation of $h : P\,[a/b]$

From the descriptions given in the tables above, it is easy to see how a successful simulation of a rule $h : P\,[x_h/y_h] \in P$ works. If we enter a membrane $(h,1)$ with a string v not being of the form ux_h, then at some moment the only chance will be to use $F+$, introducing the trap symbol F which cannot be erased anymore and definitely leads to a non-halting computation. The additional symbols D, E, H, J, M intermediately introduced on the right-hand side of the string guarantee that loops inside the linear membrane structure for the simulation of a rule $h : P\,[x_h/y_h] \in P$ cannot lead to successful computations as well. In sum, we conclude $L\,(\Pi) = L\,(G)$. □

Due to the matrix-like membrane structure of the simple P systems constructed in the preceding proof, we could obtain the computational completeness of matrix grammars of type D^1I^1 as an obvious consequence of Theorem 5, yet the direct transformation of the construction given in the proof of this theorem would yield a lot of matrices with lengths more than 3, whereas the direct proof given in Theorem 3 only needed matrices of length at most 3.

4 Conclusion

In this paper we have considered string rewriting systems using the operations of minimal left and right insertion and deletion. Using even only the operations of minimal left insertion and minimal right deletion, matrix grammars reach computational completeness with matrices of length at most 3; our conjecture is that this required length cannot be reduced to 2. As our main result, we have shown that sequential P systems using the operations of minimal left and right insertion and deletion are computationally complete, thus solving an open problem from [2]. The simple P system constructed in the proof of Theorem 5 had rather large tree height; it remains an open question to reduce this complexity parameter. On the other hand, in Theorem 4 we have shown that using minimal left insertion, minimal right deletion, and, in addition, minimal right mutation (substitution of one symbol by another one on the right-hand side of a string) we can reduce the height of the tree structure of the P system to the minimum 1 and even avoid the use of the target *here*. Moreover, we would also like to avoid the target *here* in the case of simple P systems using minimal left and right insertion and deletion, as with avoiding the target *here*, the applications of the rules could be interpreted as being carried out when passing a membrane, in the sense of a molecule passing a specific mebrane channel from one region to another one. We shall return to this qestion and related ones in an extended version of this paper.

References

1. A. Alhazov, A. Krassovitskiy, Y. Rogozhin, S. Verlan: P Systems with minimal insertion and deletion. *Theor. Comp. Sci.* **412** (1-2), 136–144 (2011).
2. A. Alhazov, A. Krassovitskiy, Y. Rogozhin, S. Verlan: P Systems with insertion and deletion exo-operations. *Fundamenta Informaticae* **110** (1-4), 13–28 (2011).

3. A. Alhazov, A. Krassovitskiy, Y. Rogozhin: Circular Post machines and P systems with exo-insertion and deletion. In: M. Gheorghe et al.(eds.): *Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers*, LNCS **7184**, Springer, 73–86 (2012).
4. R. Benne: RNA Editing: The Alteration of Protein Coding Sequences of RNA. Ellis Horwood, Chichester, West Sussex, 1993.
5. F. Biegler, M. J. Burrell, and M. Daley: Regulated RNA rewriting: Modelling RNA editing with guided insertion. *Theor. Comput. Sci.* **387** (2), 103–112 (2007).
6. E. Csuhaj-Varjú, A. Salomaa: Networks of parallel language processors. In: Gh. Păun, A. Salomaa (eds.): *New Trends in Formal Languages.* LNCS **1218**, 299–318, Springer, Heidelberg (1997).
7. J. Castellanos, C. Martín-Vide, V. Mitrana, J.M. Sempere: Solving NP-complete problems with networks of evolutionary processors. In: J. Mira, A.G. Prieto (eds.): *IWANN 2001.* LNCS **2084**, 621–628, Springer, Heidelberg (2001).
8. J. Dassow, F. Manea, B. Truthe: On normal forms for networks of evolutionary processors. *Proc. UC 2011*, 89–100 (2011).
9. J. Dassow, F. Manea: Accepting hybrid networks of evolutionary processors with special topologies and small communication. *Proc. DCFS 2010*, 68–77 (2010).
10. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, 1989.
11. R. Freund, M. Kogler, Y. Rogozhin, S. Verlan: Graph-controlled insertion-deletion systems. In: I. McQuillan, G. Pighizzini (eds.): *Proc. of 12^{th} Workshop on Descriptional Complexity of Formal Systems, vol. 31 of EPTCS*, 88–98 (2010).
12. R. Freund, M. Oswald, A. Păun: Gemmating P systems are computationally complete with four membranes. In: L. Ilie, D. Wotschke: *Pre-proceedings DCFS 2004.* The University of Western Ontario, Rep. No. 619, 191–203 (2004).
13. B. Galiukschov: Semicontextual grammars. *Logica i Matem. Lingvistika*, 38–50. Tallin University (in Russian) (1981).
14. D. Haussler: *Insertion and Iterated Insertion as Operations on Formal Languages.* PhD thesis, Univ. of Colorado at Boulder, 1982.
15. D. Haussler: Insertion languages. *Information Sciences* **31** (1), 77–89 (1983).
16. S. Ivanov, S. Verlan: Random context and semi-conditional insertion-deletion systems. *arXiv*, CoRR abs/1112.5947 (2011).
17. L. Kari: *On Insertion and Deletion in Formal Languages.* PhD thesis, University of Turku, 1991.
18. L. Kari, G. Păun, G. Thierrin, S. Yu: At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. In: *Proc. of 3rd DIMACS Workshop on DNA Based Computing,* 318–333, Philadelphia (1997).
19. A. Krassovitskiy, Y. Rogozhin, S. Verlan: Computational power of insertion-deletion (P) systems with rules of size two. *Natural Computing* **10** (2), 835–852 (2011).
20. S. Marcus: Contextual Grammars. *Rev. Roum. Math. Pures Appl.* **14**, 1525–1534 (1969).
21. M. Margenstern, V. Mitrana, M. Pérez-Jiménez: Accepting hybrid networks of evolutionary systems. *Proc. DNA10*, LNCS **3384**, 235–246 (2005).
22. M. Margenstern, G. Păun, Y. Rogozhin, S. Verlan: Context-free insertion-deletion systems. *Theor. Comput. Sci.* **330** (2), 339–348 (2005).
23. M.L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
24. G. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, 2002.

25. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
26. G. Păun, G. Rozenberg, A. Salomaa: DNA Computing: *New Computing Paradigms*. Springer, 1998.
27. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages,* 3 volumes. Springer-Verlag, Berlin, Heidelberg, New York, 1997.
28. I. Petre, S. Verlan: Matrix insertion-deletion systems. *arXiv*, CoRR abs/1012.5248, 2010.
29. The P systems Web page. `http://ppage.psystems.eu/`
30. S. Verlan: Recent developments on insertion-deletion systems. *Comp. Sci. J. of Moldova* **18** (2), 210–245 (2010).
31. S. Verlan: *Study of language-theoretic computational paradigms inspired by biology.* Habilitation thesis, University of Paris Est, 2010.
32. S. Verlan: On minimal context-free insertion-deletion systems. *J. of Automata, Languages and Combinatorics* **12** (1-2), 317–328 (2007).

Simulating Large-Scale ENPS Models by Means of GPU

Manuel García–Quismondo[1], Ana Brânduşa Pavel[2], and Mario J. Pérez–Jiménez[1]

[1] Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n. 41012 Sevilla, Spain
E-mail:{mgarciaquismondo,marper}@us.es

[2] Department of Automatic Control and Systems Engineering, Politehnica University of Bucharest
Splaiul Independenţei, Nr. 313, sector 6, 090042, Bucharest, Romania
E-mail: apavel@ics.pub.ro

Summary. Enzymatic Numerical P Systems (ENPS), an extension of Numerical P Systems, have been successfully applied to model robot controllers. GPGPU is an innovative technological paradigm which applies the parallel architecture of graphic cards to solve parallel, general–purpose problems. In previous work, a GPU simulator for ENPS was introduced. In this paper, a performance analysis on the simulator is performed in order to experimentally measure the speed-up factors resulting from the simulations.

Keywords: Enzymatic Numerical P Systems, GPU, simulation

1 Enzymatic Numerical P Systems

Membrane computing is an interdisciplinary field which studies computational models inspired by the compartmental structure of biological cells. There exist many types of membrane systems [12], also known as P systems after mathematician Gh. Păun, who introduced them. Numerical P systems (NPS) are a type of P systems in which numerical variables evolve inside the compartments by means of programs; a program (or rule) is composed of a production function and a repartition protocol [11]. The variables have a given initial value and the production function is a multivariate polynomial. The value of the production function for the current values of the variables is distributed among variables in certain compartments according to a repartition protocol. A formal definition of NPS can be found in [11], where this type of P system is introduced with possible applications in economics.

Enzymatic numerical P systems (ENPS) represent an extension of NPS, proposed and used in the context of modeling robot controllers [13], [14]. ENPS is a more powerful modelling tool than NPS, as it is proven in several articles [13], [3], [17]. ENPS models allow the existence of more than one rule per membrane than NPS while keeping the deterministic behavior. By using a special type of variables referred as enzymes, ENPS models provide a selection mechanism of the active rules during the computational process. Therefore, ENPS are a flexible and efficient modelling framework that can be successfully used for modeling robot behaviors like obstacle avoidance, localization, follower, etc. [13], [4]. An ENPS model of degree $m, m \geq 1$ is formally defined as follows:

$$\Pi = (H, \mu, (Var_1, E_1, Pr_1, Var_1(0)), \ldots, (Var_m, E_m, Pr_m, Var_m(0))) \tag{1}$$

where:

- H is an alphabet that contains m symbols (the labels of the membranes);
- μ is a membrane structure;
- Var_i is the set of variables from compartment i, and the initial values for these variables are $Var_i(0)$;
- E_i is a set of enzyme variables from compartment i, $E_i \subseteq Var_i$
- Pr_i is the set of programs (rules) from compartment i. Programs process variables and have two components; a production function and a repartition protocol. In ENPS models, programs can have one of the two following forms:

  1. Non-enzymatic form, which is exactly like the one from standard NPS:

  $$Pr_{j,i} = (F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}), c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}) \tag{2}$$

  2. Enzymatic form

  $$Pr_{j,i} = (F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}), e_{j,i}, c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}) \tag{3}$$

  where:
  – $F_{j,i}(x_{1,i}, ..., x_{k_i,i})$ is the production function;
  – $e_{j,i} \in E_i$, is the enzyme–like variable associated to $e_{j,i}$;
  – k_i represents the number of variables in membrane i;
  – $c_{j,1}|v_1 + \ldots + c_{j,n_i}|v_{n_i}$ is the repartition protocol;
  – n_i represents the number of variables contained in membrane i, plus the number of variables contained in the parent membrane of i, plus the number of variables contained in the children membranes of i.

In ENPS models, all active rules are executed in parallel on each computational step. A rule is always active if it is in the non-enzymatic form. Otherwise, if it is in the enzymatic form, a rule is active only if the associated enzyme–like variable has a greater value than the minimum of the absolute values of the variables involved in the production function. The repartition protocol works like in classical NPS [11]

Both NPS and ENPS models have been successfully used for modelling robot controllers [4], [13], [14]. The advantages of using ENPS for this kind of applications are pointed out in [3]. For testing the robot controllers, a Java implementation of a numerical P systems simulator was implemented and used. This Java simulator, *SimP*, simulates ENPS models. Since ENPS are an extension of NPS, SimP can be used to simulate classical NPS as well. SimP was proposed in [15] and it is available as a free executable version (for a free executable version of SimP, please contact `anabrandusa@gmail.com`). SimP allows the computation of rational production functions and not only polynomials. This is useful for implementing more complex models required for robotics applications. An example of model of a rational production function is presented in subsection 4.3.

In this paper, the performance of a parallel and distributed simulator which computes ENPS structures is analysed. The simulator was first proposed in [7]. In order to test the parallel simulator and analyse its performance, a replication mechanism of membranes was used. For instance, in swarm robotics applications, the robots in a group may have similar behaviors which need to run in parallel. For example, each robot needs to run an obstacle avoidance behavior. Therefore, the ENPS structure for obstacle avoidance [13] must be executed in parallel for all the robots in the swarm (figure 1). Those applications in which each robot needs to run one or even more membrane controllers in parallel and all robots must work in parallel as well take real advantage of the parallel and distibuted ENPS simulator, which will be further discussed.

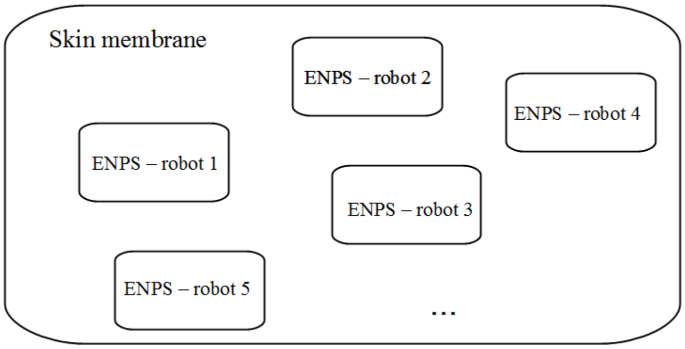


Fig. 1. Replication of ENPS models can be used for swarm robotics applications

2 An introduction to GPU Computing

Graphic cards, also known as Graphic Processing Units (*GPUs*) are devices whose main task is to solve image rendering problems. These problems are usually massive parallel problems, as they can commonly be reduced to render pixels and vertices in a parallel fashion. As a result, the industry has turned these devices into powerful highly–parallel computers with a large number of processors. However, they have been of limited use to scientist for quite a long time. The reason is that GPUs were only suitable for image–related problems, with little application in the scientific world out of image processing itself. Therefore, the amount of effort required to translate general–purpose problems into their graphical interpretations made scientists use other parallel architectures such as computer clusters and FPGAs [1] [9] [16].
However, NVIDIA changed this landscape by providing a toolkit for general–purpose GPU computing named Compute Unified Device Architecture (*CUDA*) [19]. This API permitted developers to solve scientific, non–graphical problems on GPUs. From then on, the adoption of GPU computing by the scientific community has gone widespread. As a result numerous scientific papers have shown moderate to impressive performance improvements on a GPU over a CPU [2].

2.1 The CUDA programming model

Nowadays, the number of processors in a GPU can reach up to 448 processor cores and 1.536 processing units per core, thus resulting in a total number of $448 \times 1.536 = 688.128$ processing units [19]. Thus, GPUs are structured as a grid of multiprocessor cores or *streaming multiprocessors.* Each one of these multiprocessors (figure 2) is composed of several simpler processing units known as *streaming processors.* This hierarchy allows programmers to structure their code in a distributed way, so they can couple processes with a high communication bandwidth with each other, thus clustering and allocating them within the same multiprocessor in the GPU. All processing units execute the same code at the same time, hence applying a computational paradigm known as Single Instruction Multiple Program (*SIMP*) [16]. CUDA provides an abstract model of GPU architecture. This model is known as the *CUDA programming model.* The idea is to make use of an abstraction of the specific graphic card on which the code runs, so as not to force the code depend on particular devices. This model is composed of a grid of elements known as *blocks*, which are abstractions of the streaming multiprocessors mentioned above. Each one of these blocks is composed of computing elements known as *threads* [5]. Thus, the CUDA programming model is a multidimensional three–levelled architecture , as the dimension of grids and blocks can be 1, 2 or 3, depending on the configuration options set by the developer. Figure 3 describes graphically this programming model.

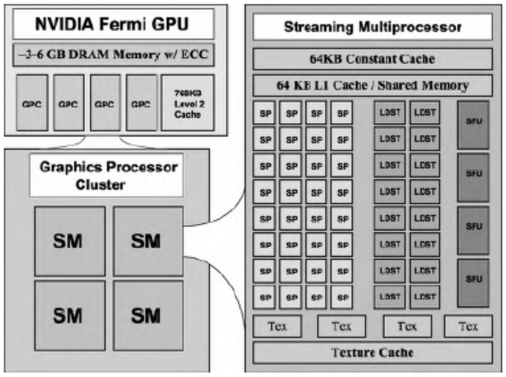


Fig. 2. Simplified hardware block diagram of an Nvidia Graphic Card

2.2 Programming on CUDA–C

The firsts steps towards obtaining languages for general–purpose GPU computing involved wrapping currently existing languages for graphics processing with mainstream general–purpose languages, such as C and Fortran. However, these approaches were still hindered by the limitations of graphic cards on treating general–purpose data structures and data flows as pixels and vertices. However, the CUDA programming

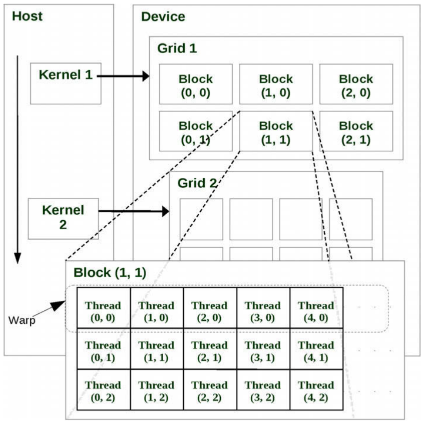


Fig. 3. The CUDA programming model

model, combined with more advanced graphic hardware with support for general–purpose parallel programming allowed to extend mainstream languages with specific primitives for GPU computing [16]. In this direction, Nvidia issued the first version of the *CUDA–C* language in 2006 [19]. This language was an extension of the C language with primitives and operations for general–purpose parallel problems [16].

A CUDA–C program is composed of two different parts: the *host* code and the *device* code. The host code is the part of the code executed on the CPU. This code contains calls to pieces of the device code, which is executed on the GPU. The device code is composed of CUDA–C functions which are executed on the GPU, known as *kernels*. The SIMP paradigm defines that each kernel is executed on all threads at the same time. When it comes to execution, threads are bundled in packages known as *warps*. Threads in the same warp communicate via a fast on–chip memory. However, the communication with threads in different warps is performed by using a slow off–chip memory. That is the reason why threads in the same warp should have a high communication bandwith with each other, as well as a low communication bandwith with threads out of the warp [5].

In practice, the CUDA programming model claims that each block is assigned a sequence of warps. Therefore, each block takes a warp from its sequence and executes it, assigning each warp thread to a different block thread [16]. The order of this sequence, as well as the block to which each warp is associated, is not controlled by the developer. This entails that the problem should not depend on the order in which the warps are executed, as this order cannot be guaranteed [5]. A more thorough description of the CUDA programming model can be found in [10].

3 A GPU Simulator for Enzymatic Numerical P Systems

3.1 A brief description of the simulator

In a previous work [7], a GPU simulator for Enzymatic Numerical P Systems was introduced. This simulator takes as input an XML file defining a description of an ENPS model and simulates it for a number of cycles defined. This number of cycles can be defined in the XML file or as an optional argument in the call sentence. The simulator does not need to check for errors in the input file. The reason is that the XML format accepted by the simulator is also accepted by SimP, which is a Java simulator for ENPS proposed in [15]. This way, errors in the specification of the ENPS model can be checked on SimP. Therefore, if a specification is regarded as error–free by SimP, then it can be given as input to the GPU simulator. This simulator will be published under GNU GPL version 3 license [8], and it is currently available by contacting the authors. In [7], some open problems were proposed. One of this problem had to do with the comprehensive evaluation of the performance of the simulator on large–scale models composed of a considerably high number of programs. In these cases, the overheads related to the workload distribution and the setup operations needed to run the simulations were supposed to be minimal, in comparison to the speed–up factor obtained from the parallel application of the programs in the model. This is not the case of models with a small numbers of programs. In these cases, the performance gain obtained as a result of the parallel application of programs is suffocated by the considerably time–consuming task of setting up the CUDA programming model elements [7].

3.2 Functioning of the GPU simulator

The functioning of the GPU simulator described in [7] consists of two stages. The first stage initializes the model to simulate. The second stage simulates a computational step of the model. This stage is repeated for each computational step simulated. These stages are described below:

Initialization stage: First, all operations concerning the setup of the GPU device itself are carried out. These operations include allocations in the GPU memory and transactions between the CPU memory and the GPU memory. Then, this stage normalizes the coefficients found in the repartition protocols in the input model. In practical terms, this normalization substitutes each coefficient $c_{l,s}$ by $\frac{c_{l,s}}{\sum_{j=0}^{k_{l,i}} c_{l,j}}$. This normalization can be performed on each simulation step, but performing it only once at the beginning of the simulation spares computational time.

Computation stage: This stage simulates checks the programs in the model and applicates the checked programs. It consists of four sub–steps, which are:

1. For each program, set its activation. That is, check whether this program is active, that is, is going to be applied on the current computational step.
2. Calculate the production function of the active programs.
3. Set to 0 the values consumed by the active programs, that is, the values of variables such that there is any active program which depends on its value.

4. Multiply the results of the active production functions by the normalized coefficients calculated on the initialization stage.
5. Add these results to the variables contributed by the active programs, according to their repartition protocols.

4 Performance analysis of the simulator

In order to carry out an analysis of the performance and runtimes obtained from the GPU simulator, we have developed a sequential simulator for Enzymatic Numerical P Systems in C language. This sequential simulator has been developed to compare simulation times obtained from a sequential, low–level simulator to those obtained from the GPU simulator. Nevertheless, it can also be used for the efficient simulation of Enzymatic Numerical P Systems in those environments in which no Nvidia card is available. Besides, this simulator takes as input a file which describes an ENPS in the same format that the GPU simulator and SimP. Therefore, the same files can be used for all three simulators, hence sparing time on translations between formats.
For a fair comparison between execution times, no memory allocation is performed after the setup stage. This feature is compulsory on the GPU simulator, because all computation steps are implemented by means of kernel calls and all memory in the GPU can only be allocated from the host code [19]. The importance of this feature rises from the fact that memory allocation in C is a time–consuming instruction. Therefore, if there were a significant number of memory allocations on each computational step the performance of the C sequential simulator would be severely hindered. This would result on an even higher speed–up factor due to a bad design of the sequential simulator, instead of a good design of its GPU counterpart.

4.1 Performance comparison with SimP

Apart from comparing the CUDA–C simulator with a C–based one, we have also compared the GPU simulation times with the ones obtained from SimP [15]. SimP is an ENPS simulator in Java language. Java programs are executed on a virtual machine which does as a middleware between the actual device and the software. This virtual machine is known as Java Virtual Machine (JVM) [18]. JVM ensures that Java programs can be executed on any device in which JVM is installed, thus guaranteeing complete compatiblity among different hardware architectures. However, this virtual machine approach comes at a cost. Firstly, the programmer loses control of the way in which the memory is managed. For instance, memory objects cannot be freed directly. Instead, an execution thread named *garbage collector* checks which objects are not referenced anymore in the program and frees the allocated memory. Secondly, the translatiosn from JVM instructions to assembly instructions are performed in runtime. Thus, an overhead in the execution time is produced as a result of these translations. All in all, the programmer cannot control directly the execution flow of Java programs. Therefore, in cases where efficiency is required, Java is not, in most cases, a true rival to low–level languages such as C.

4.2 Generation of input models

In [7], some problems about an extensive analysis of the simulator are discussed. One of them has to do with the fact that the existing ENPS models have too few programs for parallel simulations to pay off. The reason is that the setup operations computed at the beginning of parallel simulations take a long time, in comparison to the whole sequential computation runtime. In order to overcome this difficulty, we have taken some ENPS models as reference and replicated them several times. This way, we can have an on–demand number of programs per model. Therefore, the more times the seed models are replicated, the more programs will compose the resulting models. When the number of replications given as input is high enough, the GPU simulation does pay off in terms of execution time.
The algorithm used for performing the model replication takes the following inputs:

N: The number of copies to perform.
Π: The seed model to replicate:
$\Pi = (H, \mu, (Var_1, Pr_1, E_1, Var_1(0)) \dots (Var_m, Pr_m, E_m, Var_m(0)))$.
For these inputs, the algorithm takes the following steps:

1. Replicate Π a number of times N.
2. Associate an index $o, 1 \leq o \leq N$ to each of the copies of Π. As a result, a set Φ of ENPS models of degree m is obtained. Formally speaking, $\Phi = \{\Pi_o = (H_o, \mu_o, (Var_{1,o}, E_{1,o}, Pr_{1,o}, Var_{1,o}(0)) \dots (Var_{m,o}, E_{m,o}, Pr_{m,o}, Var_{m,o}(0)))\}$, $1 \leq o \leq N$, where:
   - $\forall o | 1 \leq o \leq N, H_o = \{\{1, o\} \dots \{m, o\}\}$
   - $\forall i, o | 1 \leq i \leq m, 1 \leq o \leq N, Var_{i,o} = \{x_{1,i,o} \dots x_{k_i,i,o}\}$
   - $\forall i, o | 1 \leq i \leq m, 1 \leq o \leq N, E_{i,o} \subseteq Var_{i,o}$ is the set of all enzyme–like variables associated to programs in $Pr_{i,o}$.
   - $\forall i, o | 1 \leq i \leq m, 1 \leq o \leq N, Pr_{i,o} = Pr_{1,i,o} \dots Pr_{q_i,i,o}$, where:
     - $Pr_{l,i,o} = (F_{l,i,o}(x_{1,i,o}, \dots, x_{k_i,i,o}) \rightarrow c_{l,1,o}|v_{o,1} + \dots + c_{l,n_i,o}|v_{o,n_i})$ if $Pr_{l,i}$ is in non–enzymatic form.
     - $Pr_{l,i,o} = (F_{l,i,o}(x_{1,i,o}, \dots, x_{k_i,i,o})(e_{l,i,o} \rightarrow) c_{l,1,o}|v_{o,1} + \dots + c_{l,n_i,o}|v_{o,n_i})$ if $Pr_{l,i}$ is in enzymatic form.
   - $\forall i, o | 1 \leq i \leq m, 1 \leq o \leq N, Var_{i,o}(0) = \{\lambda_{1,i,o} \dots \lambda_{k_i,i,o}\}$
3. $\forall i, o, j, l, (1 \leq i \leq m), (1 \leq o \leq N), (1 \leq j \leq k_i), (1 \leq l \leq q_i)$, assign a different random value in $\{1, \dots, 10\}$ to $\lambda_{j,i,o}, c_{l,n_i,o}$ and each one of the numerical constants in $F_{l,i,o}$.
4. Return a new ENPS model $\Pi_r = (H_r, \mu_r, (Var_{1,1}, E_{1,1}, Pr_{1,1}, Var_{1,1}(0)) \dots (Var_{m,N}, E_{m,N} Pr_{m,N}, Var_{m,N}(0)), (Var_{skin}, E_{skin}, Pr_{skin}, Var(0)_{skin}))$, where:
   - $H_r = \cup_{o=1}^{N} H_o \cup \{skin\}$
   - $\mu_r = [\mu_1 \dots \mu_N]_{skin}$
   - $Var_{skin} = \emptyset$
   - $E_{skin} = \emptyset$
   - $Pr_{skin} = \emptyset$
   - $Var(0)_{skin} = \emptyset$

The replication process has been performed by using Java [18]. For the purposes of parsing the seed model and writing the resulting models, an extension of P–Lingua [6] has been developed. Specifically, a new input and a new output format has been included into the P–Lingua framework. The input format delegates the parsing process to SimP [15]. The output format generates an XML description of the model. This description is encoded on the common format accepted by all ENPS simulators (Java, C and CUDA–C). This extension proves the versatility of P–Lingua as a useful, assisting tool for a wide variety of Membrane Computing–related tasks; in our case; for analysing the performance of a GPU simulator.

4.3 Case study

In our simulator, two seed models have been replicated. After these replications, the resulting expanded models have been simulated. The first seed model is a dummy one with no particular purpose apart from this performance analysis. This model is an ENPS composed of 2 membranes: $\Pi_1 = (H, \mu, (Var_1, E_1, Pr_{1,1}, Var_1(0)), (Var_2, E_2, Pr_{1,2}, Var_2(0)))$, where:

- $H = 1, 2$
- $\mu = [[]_2]_1$
- $Var_1 = \{x_{1,1}, x_{2,1}, x_{3,1}\}$
- $E_1 = \{x_{3,1}\}$
- $Pr_{1,1} = \{3 \cdot x_{1,1}(x_{3,1} \rightarrow)2|x_{1,1} + 1|x_{2,1}\}$
- $Var_1(0) = \{1, 2, 3\}$,
- $Var_2 = \{x_{1,2}\}$
- $E_2 = \emptyset$
- $Pr_{1,2} = \{2 \cdot x_{1,2} \rightarrow 2|x_{1,2}\}$
- $Var_2(0) = \{1\}$

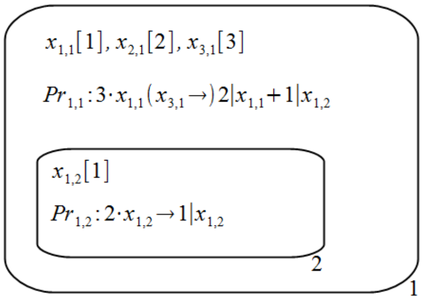

$x_{1,1}[1], x_{2,1}[2], x_{3,1}[3]$
$Pr_{1,1}:3\cdot x_{1,1}(x_{3,1}\rightarrow)2|x_{1,1}+1|x_{1,2}$
$x_{1,2}[1]$
$Pr_{1,2}:2\cdot x_{1,2}\rightarrow 1|x_{1,2}$
2
1

Fig. 4. Dummy ENPS used as seed for replication

On the other hand, the second seed model performs a function approximation. Mathematical functions like trigonometric functions, exponential functions, etc. are often used in control algorithms in robotics. Therefore, in the following example an ENPS model which computes e^x is presented. The proposed GPU simulator also allows computation of rational production functions as well as polynomial functions, which is an important advantage for the modeling process of complex membrane systems. In order to approximate e^x, the following power series is used:

$$e^x \approx \sum_{n \geq 0} \frac{x^n}{n!} \tag{4}$$

The partial sum of this power series is the next sequence:

$$s_n = \sum_{k \geq 0}^{n} \frac{x^k}{k!} \tag{5}$$

Sequence s_n can be written in a recurrent form, as follows:

$$s_n = s_{n-1} + a_n \tag{6}$$

where a_n is:

$$a_n = \frac{x^n}{n!} \tag{7}$$

Sequence a_n can also be written in a recurrent form, by computing $\frac{a_n}{a_{n-1}}$ as follows:

$$a_n = \frac{x}{n} \cdot a_{n-1} \tag{8}$$

Formula 8 can be implemented as a rational production function, as shown in figure 5 (rule $Pr_{1,2}$).

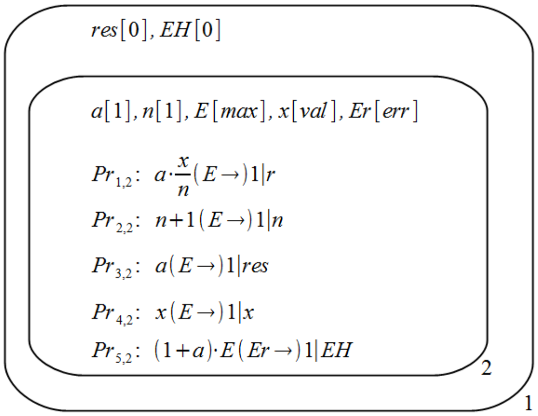

$res[0], EH[0]$
$a[1], n[1], E[max], x[val], Er[err]$
$Pr_{1,2}: \; a \cdot \frac{x}{n}(E \rightarrow) 1|r$
$Pr_{2,2}: \; n+1(E \rightarrow) 1|n$
$Pr_{3,2}: \; a(E \rightarrow) 1|res$
$Pr_{4,2}: \; x(E \rightarrow) 1|x$
$Pr_{5,2}: \; (1+a) \cdot E(Er \rightarrow) 1|EH$
2
1

Fig. 5. An ENPS which computes e^x. This model has also been used as seed model for replication

As it is shown in figure 5, two membranes were used in order to approximate the exponential function e^x. The skin membrane (membrane 1) contains a non enzyme–like

variable res which represents the result of the computation and an enzyme–like variable, EH, which is a stop enzyme. Stop enzymes are used in order to test the stop condition of the computation. Therefore, the number of computational steps is different for different arguments of the function. The computation finishes when the value of the term added to the sum is lower than a given value, err.
The child membrane (membrane 2) is responsible for the approximation. It contains the following variables:

- a stores the next term of the a_n sequence and has an initial value equal to 1.
- n is a counter variable and has the initial value equal to 1.
- x represents the argument of the function e^x.
- E is an enzyme–like variable which controls the program flow. it allows the execution of the valid production functions and it is consumed when the computation finishes; the initial value of the enzyme is given as input max; max must be a value grater than the maximum possible value of x.
- Er is an enzyme–like variable used in the stop condition.

Er receives an input value, $err = 10^{-10}$; when the term of the series is lower than err, the computation stops.
Membrane 2 has five rules responsible for the following tasks:

- $Pr_{1,2}$ computes the next term in the series; if the value of E is greater than a, x or n, the rule is active.
- $Pr_{2,2}$ produces the incrementation of n.
- $Pr_{3,2}$ accumulates the terms in variable res, which will be the final result.
- $Pr_{4,2}$ copies the value of x, which was consumed and must be stored.
- $Pr_{5,2}$ stops the computation when $a < Er$.

The value of a is decreasing because the sequence a_n is convergent. When $Pr_{5,2}$ is activated, the stop enzyme EH, receives a positive value and E is consumed, so the other production functions become inactive. A condition outside the membrane system tests if the stop enzyme, EH, is greater than 0 and if that happens, the simulation stops.

4.4 Simulation results

For each seed, a total of 36 models have been simulated. The number of programs range from 1 to 9000. Each model has been run for 100 steps. Figure 11 displays the execution times obtained from the Java simulator included in SimP [15], the C simulator described in section 4 and the CUDA–C simulator from [7]. Figure 13 displays the speed–up factors obtained from the runtimes among the simulators. Figures 14 and 12 plot the same data, with the exception of the Java results. The reason is to display cleaner statistics on the execution times and speed–up factors obtained from the most efficient studied simulators. By examining the dummy model charts, one can observe that there is a great difference between the execution times from the SimP (Java) simulator and the C and CUDA–C simulators. Thus, a maximum speed–up factor of about 85x is reached on the comparison between the CUDA–C and Java simulators. However, the maximum speed–up factor obtained between the CUDA–C and Java simulators is only 6x. The maximum speed–up factor on the simulation of the e^x model is about 49x on the Java vs CUDA–C comparison and about 10x on the C vs CUDA–C comparison. The used graphic card is a domestic, commercial one. Thus, it is not designed for intensive parallel, computations. In contrast, Tesla models contain more RAM memory and a larger number of processors, as they are specifically engineered for intensive High Performance Computing [19]. For instance, Nvidia Tesla C1060 graphic cards contain 240 streaming multiprocessors and 4GB RAM memory [19]. Thus, the speed–up factors obtained on one of these cards is expected to be higher than in the ones obtained in this study. An extension of this performance analysis on these extensive cards is left for future work.

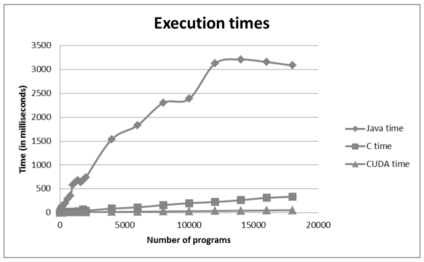


Fig. 6. Execution times for SimP (Java), C and CUDA-C ENPS simulators

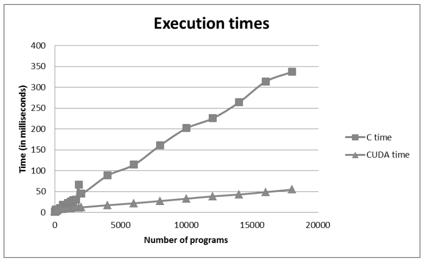


Fig. 7. Execution times for C and CUDA-C ENPS simulators

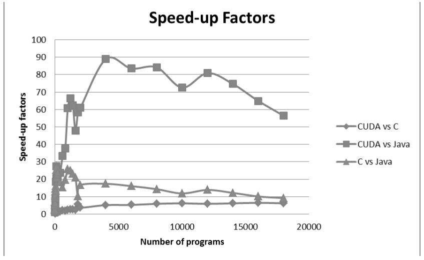


Fig. 8. Speed–up factors for SimP (Java), C and CUDA-C ENPS simulators

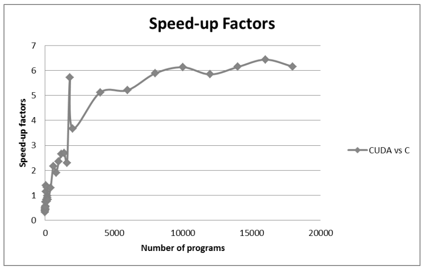


Fig. 9. Speed–up factors for C and CUDA-C ENPS simulators

Fig. 10. Execution times and speed–up factors obtained from the simulation of the dummy model

The models have been simulated on an *Nvidia GeForce GTX 460M* card with 1.5GB of dedicated RAM memory [19]. This model supports the new Fermi technology. Fermi cards allow programmers to make use of new features impossible (or at least very hard) for previous models. Some examples of these features are the computation of recursive functions and atomic operations with float–type numbers [19]. The impact of these features on the simulator code is really important. Thus, by employing these features, the development process is eased and the simulator code is much clearer. For instance, in order to calculate production functions in programs, recursion comes as a straightforward approach. That is because these general mathematical expressions can be easily represented as tree–like structures. Tree–like structures are usually stepped through by using recursive algorithms. Another important feature brand–new on Fermi technology

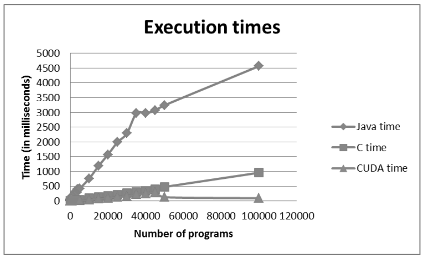


Fig. 11. Execution times for SimP (Java), C and CUDA-C ENPS simulators

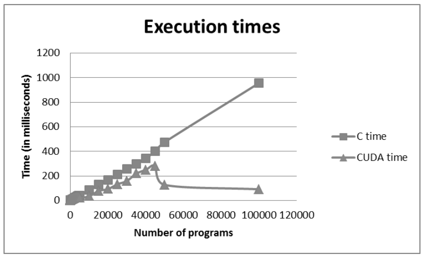


Fig. 12. Execution times for C and CUDA-C ENPS simulators

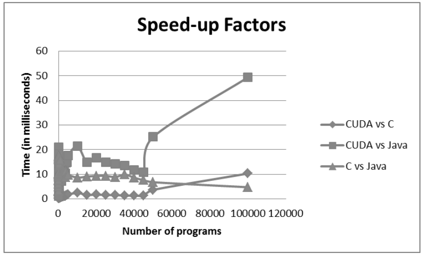


Fig. 13. Speed–up factors for SimP (Java), C and CUDA-C ENPS simulators

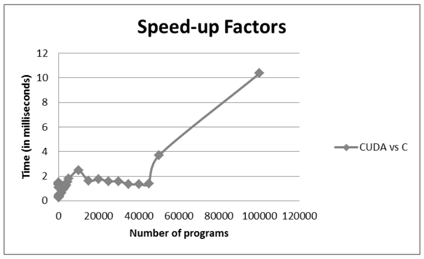


Fig. 14. Speed–up factors for C and CUDA-C ENPS simulators

Fig. 15. Execution times and speed–up factors obtained from the simulation of the e^x model

is the use of atomic operations on floating–point numbers. In order to add up the contributions from repartition protocols, these instructions have been used, as the values of contributed variables can be modified by different CUDA threads. However, the use of these features comes as a cost. Specifically, the GPU simulator can only be run on Fermi Nvidia cards, as previous models do not support these features. An improvement on the code in order to add compatibility for previous graphic cards is thus left as a future work.

5 Conclusions

This paper displays how much the simulation of ENPS models can be accelerated by applying the GPU technology. It shows a noticeable speed–up factor obtained from comparing the CUDA–C and C simulation runtimes and a dramatic acceleration when these execution runtimes are compared with a previously existent Java simulator. As the number of processors in the GPU device and the number of programs per model is augmented, one can expect a greater speed–up factor. The Research Group on Natural Computing owns a High Performance Computing server equipped with an Nvidia Tesla C1060 card [19], [5]. These cards contain 240 streaming multiprocessors and 4GB of RAM memory. Unfortunately, our simulator is unable to run on this server. The reason is that

Nvidia Tesla C1060 cards do not implement the new Fermi technology. Therefore, they cannot execute recursive functions and atomic operations on floating–point numbers. Hence, an adaptation of our code to lay out these constraints is conveniently left as a future extension.

6 Acknowledgements

Manuel García-Quismondo and Mario J. Pérez-Jiménez are supported by project TIN 2009-13192 from "Ministerio de Ciencia e Innovación" of Spain, co-financed by FEDER funds. Manuel García-Quismondo and Mario J. Pérez-Jiménez are also supported by "Proyecto de Excelencia con Investigador de Reconocida Valía P08-TIC-04200" from Junta de Andalucía. Manuel García-Quismondo is also supported by the National FPU Grant Programme from the Spanish Ministry of Education.

References

1. N. Azizi, I. Kuon, A. Egier, A. Darabiha, P. Chow. Reconfigurable molecular dynamics simulator. *Annual IEEE Symposium onf Field–Programmable Custom Computing Machines*, 2004, 197–206.
2. F. Bleichrodt, R.H. Bisseling, H.A. Dijkstra. Accelerating a barotropic ocean model using a GPU. *Ocean Modelling*, 41, (2012), 16–21.
3. C. Buiu, A.B. Pavel, C.I. Vasile, I. Dumitrache. Perspectives of using membraine computing in the control of mobile robots. *Beyond AI. Interdisciplinary Aspects of Artificial Intelligence*, 08/12/2011-09/12/2011, Pilsen, Czech Republic.
4. Buiu, C., Vasile, C., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences, vol 187, doi:10.1016/j.ins.2011.10.007 (2012), 33-51.
5. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with Active Membranes on CUDA. *Briefings in Bioinformatics*, 11, 3 (2010), 313-322.
6. M. García-Quismondo, R. Gutiérrez-Escudero, M.A. Martínez-del-Amor, E. Orejuela-Pinedo, I. Pérez-Hurtado. P-Lingua 2.0: A software framework for cell-like P systems. *International Journal of Computers, Communications and Control*, 4, 3 (2009), 234-243.
7. M. García-Quismondo, M.J. Pérez–Jiménez, L.F. Macías–Ramos. Implementing ENPS by means of GPUs for AI applications. *Beyond AI. Interdisciplinary Aspects of Artificial Intelligence (BAI 2011)*, 08/12/2011-09/12/2011, Pilsen, Czech Republic.
8. *http://www.gnu.org/copyleft/gpl.html*
9. Y. Gu, T. VanCourt, M. Herbordt. Accelerating molecular dynamics simulations with configurable circuits. *International Conference on Field–Programable Logic and Applications*, 2005, 475–480.
10. J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with CUDA, *Queue*, 6, 2 (2008), 40-53.
11. Gh. Paun, R. Paun. Membrane Computing and Economics: Numerical P Systems. *Fundamenta Informaticae*, 73, 1-2 (2006), 213-227.
12. Gh. Păun, G. Rozenberg, A. Salomaa (eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press (2010).

13. A.B. Pavel, C. Buiu. Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, DOI: 10.1007/s11047-011-9286-5 (2011).
14. A.B. Pavel, C.I. Vasile, I. Dumitrache: Robot localization implemented with enzymatic numerical P systems. submitted.
15. A.B. Pavel, Membrane controllers for cognitive robots. *Masters thesis*, Department of Automatic Control and System Engineering, Politehnica University of Bucharest, Romania, February 2011.
16. J.E. Stone, D.J. Hardy, I.S. Ufimtsev, K. Schulten. GPU-accelerated molecular coming of age. *Journal of Molecular Graphics and Modelling*, 29, (2010), 116–125.
17. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: On the Power of Enzymatic Numerical P Systems. submitted.
18. *http://www.java.com*
19. *http://www.nvidia.com/object/cuda_home_new.html*

A Kernel P System

Marian Gheorghe[1,2], Florentin Ipate[2], and Ciprian Dragomir[1]

[1] Department of Computer Science
University of Sheffield
Portobello Street, Regent Court, Sheffield, S1 4DP, UK
{m.gheorghe, c.dragomir}@sheffield.ac.uk

[2] Department of Computer Science
University of Pitesti
Str Targu din Vale, 1, Pitesti, Romania
florentin.ipate@ifsoft.ro

Summary. A basic P system, called kernel P system (kP system for short), covering features of different P systems introduced and studied so far is defined and discussed. It is a relatively low level specification system aiming to cover features exhibited by most of the problems modelled so far using P system formalisms. A small set of rules and specific strategies to run the system step by step are presented. Some preliminary results regarding the relationships between kP systems and other classes of P systems, like neural-like P systems and P systems with active membranes, are presented. Examples illustrating the behaviour of kP systems or showing how a sorting algorithm is modelled with various classes of P systems are provided. Further developments of this class of P systems are finally briefly discussed.

1 Introduction

Different variants of P systems have been used for specifying simple algorithms [4, 2], classes of NP-complete problems [7] and other various applications [5]. More specific classes of P systems have been recently considered for modelling various distributed algorithms and problems [9]. In many cases the specification of the system investigated requires features, constraints or type of behaviour which are not always provided by the model in its initial definition. It helps in many cases to have some flexibility with modelling approaches, especially in the early stages of modelling, as it might simplify the model, shorten associated processes and clarify more complex or unknown aspects of the system. The downside of this is the lack of a coherent and well-defined framework that allows us to analyse, verify and test this behaviour and simulate the system. In this respect we engage now on defining a *kernel P system* (*kP system*, for short) that, at least for this stage, will be a low level specification language including the most used concepts from P systems. In a later stage its key features will be formally defined in an operational style and finally implemented within a model checker (SPIN [3], Maude [6]) and integrated into the P-lingua platform.

We will be working with P systems having a graph-like structure (so called, *tissue P systems*) using a set of symbols, labels of membranes, rules of various types and various strategies to run them against the multiset of objects available in each region. The rules in each compartment will be of two types, (i) *object processing rules* which transform and transport objects between compartments or exchange objects between compartments and environment and (ii) *system structure rules* responsible for changing the system's topology. Each rule has a guard, defined using activators and inhibitors in a more general way than in traditional P system classes. An execution strategy can now be specified individually, for each compartment, allowing for more complex rule selection and iteration procedures in addition to the classical maximal parallelism and sequential methods. We consider rewriting and communication rules based on promoters and inhibitors as they seem to be amongst the most flexible and general processing rules, and a special set of symport/antiport rules; additional features like membrane division, dissolution, bond creation and destruction are also considered. Two types of P systems, neural-like P systems and P systems with active membranes, are simulated by the newly introduced P systems. We analyse a specific case study based on a sorting algorithm which is described using the currently introduced model, kP systems, and some other formalisms, using electrical charges, states and labels.

2 kP Systems

A kP system is a formal framework that uses some well-known features of existing P systems and includes some new elements and, more importantly, it offers a coherent view on integrating them into the same formalism. The key elements of a kP system will be formally defined in this section, namely objects, types of rules, internal structure of the system and strategies for running such systems. Some preliminary formal concepts describing the syntax of kP systems and an informal description of the way these systems are executed will be introduced.

We consider that standard concepts like strings, multisets, rewriting rules, and computation are well-known concepts in P systems and indicate [11] as a comprehensive source of information in this respect. First we introduce the key concept of a compartment.

Definition 1. *Given a finite set, A, called* alphabet*, of elements, called* objects*, and a finite set, L, of elements, called* labels*, a* compartment *is a tuple $C = (l, w_0, R^\sigma)$, where $l \in L$ is the label of the compartment, w_0 is the initial multiset over A and R^σ denotes the DNA code of C, which comprises the set of rules, denoted R, applied in this compartment and a regular expression, σ, over $Lab(R)$, the labels of the rules of R.*

The precise format and the types of rules used in this context will be discussed in Section 2.1.

Definition 2. *A* kernel P system *of degree n is a tuple*

$$k\Pi = (A, L, IO, \mu, C_1, \ldots, C_n, i_0),$$

where A and L are, as in Definition 1, the alphabet *and the set of* labels*, respectively; IO is a multiset of objects from A, called* environment*; μ defines the membrane structure, which is a graph, (V, E), where V are vertices, $V \subseteq L$ (the nodes are labels of these compartments), and E edges; $C_1, \ldots, C_n$ are the n compartments of the system - the inner part of each compartment is called* region*, which is delimited by a* membrane*; the labels of the compartments are from L and initial multisets are over A; i_o is the* output compartment *where the result is obtained.*

As usual in P systems, the environment contains an arbitrary number of copies of each object. Each compartment is specified according to Definition 1.

2.1 kP System Rules

The discussion below assumes that the rules introduced belong to the same compartment, C_i, labelled l_i.

Each rule r may have a **guard** g, in which case r is applicable when g is evaluated to true. Its generic form is $r\ \{g\}$. The guards are constructed according to certain criteria described below. Before presenting these criteria we introduce some notations.

We consider multisets over $A \cup \bar{A}$, where A and $\bar{A}$ are interpreted as **promoters** and **inhibitors**, respectively; $\bar{A} = \{\bar{a} | a \in A\}$. For a multiset w over $A \cup \bar{A}$ and an element a from the same set we denote by $\#_a(w)$ the number of a's occurring in w. We also consider the set of well-known relational operators $Rel = \{<, \leq, =, \neq, \geq, >\}$. For a multiset $w = a_1^{n_1} \ldots a_k^{n_k}$, $a_j \in A \cup \bar{A}$, $1 \leq j \leq k$, and $\alpha_j \in Rel$, $1 \leq j \leq k$, we introduce the following notation $w' = \alpha_1 a_1^{n_1} \ldots \alpha_k a_k^{n_k}$; a_j is not necessarily unique in w or w' (as it will transpire from the explanations below, this case may occur when the multiplicity of a symbol belongs to an interval); w' is called *multiset* over $A \cup \bar{A}$ with *relational operators* over *Rel*.

If g is a guard defined according to the criteria below and *pr*, a predicate over this set of guards, then:

- $g = \epsilon$ means $pr(\epsilon)$ is always *true*, i.e., no condition is associated with the rule r; this guard is almost always ignored from the syntax of the rule;
- g is a *multiset* over $A \cup \bar{A}$ with *relational operators* over *Rel*, i.e., $g = \alpha_1 a_1^{n_1} \ldots \alpha_k a_k^{n_k}$, then $pr(w)$ is *true* iff for z, the current multiset of C_i, we have, for every $1 \leq j \leq k$, either (i) if $a_j \in A$ then $\#_{a_j}(z)\ \alpha_j\ n_j$ holds, or (ii) if $a_j \in \bar{A}$, i.e., $a_j = \bar{a}$, $a \in A$, then $\#_a(z)\ \alpha_j\ n_j$ does not hold;
- $g = w_1 | \ldots | w_p$, i.e., g is a *finite disjunction* of *multisets* over $A \cup \bar{A}$ with *relational operators* over *Rel*, then $pr(w_1 | \ldots | w_p)$ is *true* iff there exists $1 \leq j \leq p$, such that $pr(w_j)$ is *true*.

We denote by $FE(A \cup \bar{A})$, from Finite regular Expressions over $A \cup \bar{A}$ with relational operators, the set of expressions defined above. When a compound guard, cg, referring to compartments l_i and l_j is used, its generic format is $cg = l_i.g_1 \; op \; l_j.g_2$, where g_1, g_2 are finite expressions referring to compartments l_i and l_j, respectively; then, obviously, $pr(cg) = pr(g_1) \; op \; pr(g_2)$, $op \in \{\&, |\}$, where & stands for *and* and | for *or*, meaning that either both guards are true or at least one is true. Simpler forms, where one of the operands is missing, are also allowed as well as $cg = \epsilon$. A compound guard defines a Boolean condition defined across the two compartments.

Example 1. If the rule is $r : ab \to c \; \{\leq a^3 \geq b^7 = \bar{c}\}$, then this can be applied iff the current multiset consists of at most 3 a's and at least 7 b's and does not contain a single c (either none or more than 2 c's are allowed).

A rule can have one the following types:

- (a) **rewriting and communication** rule: $x \to y \; \{g\}$,
  where $x \in A^+$, $y \in A^*$, $g \in FE(A \cup \bar{A})$; the right hand side, y, has the form $y = (a_1, t_1) \ldots (a_h, t_h)$, where $a_j \in A$ and $t_j \in L$, $1 \leq j \leq h$, is an object and a target, i.e., the label of a compartment, respectively; the target, t_j, must be either the label of the current compartment, l_i, (more often ignored) or of an existing neighbour of it ($(l_i, t_j) \in E$) or an unspecified one, $*$; otherwise the rule is not applicable; if a target, t_j, refers to a label that appears more than once then one of the involved compartments will be non-deterministically chosen; if t_j is $*$ then the object a_j is sent to a neighbouring compartment arbitrarily chosen;
- (b) **input-output** rule, is a form of symport/antiport rule: $(x/y) \; \{g\}$,
  where $x, y \in A^*$, $g \in FE(A \cup \bar{A})$; x from the current region, l_i, is sent to the environment and y from the environment is brought into the current region;
- (c) **system structure rules**; the following types are considered:
  - (c1) **membrane division** rule: $[]_{l_i} \to []_{l_{i_1}} \ldots []_{l_{i_h}} \; \{g\}$,
    where $g \in FE(A \cup \bar{A})$; the compartment l_i will be replaced by h compartments obtained from l_i, i.e., the content of them will coincide with that of l_i; their labels are $l_{i_1}, \ldots, l_{i_h}$, respectively; all the links of l_i are inherited by each of the newly created compartments;
  - (c2) **membrane dissolution** rule: $[]_{l_i} \to \lambda \; \{g\}$;
    the compartment l_i will be destroyed together with its links;
  - (c3) **link creation** rule: $[]_{l_i}; []_{l_j} \to []_{l_i} - []_{l_j} \; \{cg\}$;
    the current compartment, l_i, is linked to l_j and if more than one l_j exists then one of them will be non-deterministically picked up; cg, called compound guard, describes an expression $l_i.g_1 \; op \; l_j.g_2$ as defined above;
  - (c4) **link destruction** rule: $[]_{l_i} - []_{l_j} \to []_{l_i}; []_{l_j} \; \{cg\}$;
    is the opposite of link creation and means that compartments l_i, l_j are disconnected; as usual, when more than a link, $(l_i, l_j) \in E$, exists then only one is considered by this rule; cg is a compound guard.

2.2 Regular Expressions and their Interpretation for kP Systems

In kP Systems the way in which rules are executed is described using regular expressions (over the sets of labels of rules). This approach allows the usual behaviour of P systems - requiring the rewriting and communication, and input-output rules to be applied in a maximal parallel way and structural rules (e.g. membrane division and dissolution, creation and destruction of links) to be executed one per membrane - as well as other alternative or additional features to be expressed in a consistent and elegant manner.

We first consider the set of labels of the rules, from the set R, in a given compartment, denoted by $Lab(R)$. We can define regular expressions over this set, $REG(Lab(R))$. A regular expression $\sigma \in REG(Lab(R))$ is interpreted as follows

- $\sigma = \epsilon$ means no rule from the current compartment will be executed;
- $\sigma = r$, $r \in Lab(R)$, means the rule r is executed;
- $\sigma = \alpha\beta$ means first are executed rules designed by α and then those in β;
- $\sigma = \alpha|\beta$ means either the rules designed by α or those by β are executed; often we use the notation defining sets where $|$ is replaced by ,;
- $\sigma = \gamma^*$ means rules designed by γ are executed in a maximal parallel way.

Regular expressions allows the definition of various execution strategies, including well-known maximal parallelism (and also sequential) behaviour, but also to encode more subtle concepts like order relationships between rules, which introduce a form of sequential execution. Given the above introduced types of rules we can also specify in a more coherent way the fact that maximal parallelism imposes some constraints on the way the rules dealing with the system structure are handled; it is always the case that such rules are applied one per compartment and at the end of each step. Indeed, this assumption can be made in this case as the left hand side of any of the rules c1–c4 , does not contain any object, so they are applied only when there guards are satisfied. These cases are briefly analysed below.

- Naturally, * is used to capture the maximal parallelism of a set of rules; for rules R, with $Lab(R) = \{r_1, \ldots, r_k\}$, we mostly write either $Lab(R)^*$ or $\{r_1, \ldots, r_k\}^*$, instead of $(r_1| \ldots |r_k)^*$.
- In order to express the fact that *maximal parallelism* means that *object processing rules are applied in a maximal parallel way and at the end only one of the system structure rules is applied*, we first split R, the set of rules, into R_1, containing all the object processing rules, and R_2, with all the structure defining rules, associated with the current compartment; then given the convention introduced for the set of regular expressions over $Lab(R)$, the above behaviour is expressed by $Lab(R_1)^* Lab(R_2)$.
- Now suppose that a certain order relationship exists, e.g. $r_1, r_2 > r_3, r_4$, which means that when weak priority is applied, the first two rules are executed first, if possible, then the next two. If both are executed with maximal parallelism, this is described by $\{r_1, r_2\}^*$ $\{r_3, r_4\}^*$.

These regular expressions define the strategy of executing the rules of the current compartment and together with them they form the true *DNA code* of each compartment. Other ways of executing the rules of a compartment, like equal to, less than or greater than a given number of steps, can be also considered.

The result of a computation will be the number of objects collected in the output compartment. For a kP systems $k\Pi$, the set of all these numbers will be denoted by $M(k\Pi)$.

2.3 kP System Examples

In this section we illustrate the newly introduced P system model with some examples.

Example 2. Let us consider the following kP system with $n = 4$ compartments, $k\Pi_1 = (A, L, IO, \mu, C_1, \ldots, C_4, 1)$, where
$A = \{a, b, c, p\}$,
$L = \{1, 2, 3\}$,
IO contains an arbitrary number of objects over $\{b, c\}$,
$C_1 = (1, w_{1,0}, R_1^\sigma), C_2 = (2, w_{2,0}, R_2^\sigma), C_3 = (2, w_{3,0}, R_2^\sigma), C_4 = (3, w_{4,0}, R_3^\sigma)$,
μ is given by the following graph with edges $(1, 2), (1, 3)$; $(1, 2)$ appears twice as $n = 4$ and there are two compartments, C_2, C_3, with label 2;
$w_{1,0} = a^3p$, $w_{2,0} = p$, $w_{3,0} = p$, $w_{4,0} = \lambda$, and
R_1^σ is $R_1 = \{r_1 : a \to a(b, 2)(c, 3) \ \{\geq p\}; r_2 : p \to p; r_3 : p \to \lambda\}$, and $\sigma_1 = Lab(R_1)^*$,
R_2^σ is $R_2 = \{r_1 : (b/c) \ \{\geq p\}; r_2 : p \to p; r_3 : p \to \lambda\}$, and $\sigma_2 = Lab(R_2)^*$,
R_3^σ is $R_3 = \emptyset$ and $\sigma_3 = Lab(R_3)^*$.

Please note that we do not use targets for objects meant to stay in the current compartment (i.e. we have $r_1 : a \to a(b, 2)(c, 3) \ \{\geq p\}$ instead of $r_1 : a \to (a, 1)(b, 2)(c, 3) \ \{\geq p\}$).

In this example there are only rewriting and communication rules (all the rules, but r_1 from R_2) and an input-output one (r_1 from R_2); some rules have a guard, $\geq p$ (p is a promoter), others do not have any and in each compartment the rules are applied in maximal parallel way in every step, as indicated by σ_j, $1 \leq j \leq 3$. As two instances of the compartment labelled 2, C_2, C_3, appear in the system, when the rule r_1 from the first compartment is applied, the object b goes non-deterministically to one of the two compartments labelled 2 as long as p remains in compartment 1; object c goes always to compartment labelled 3, C_4.

The initial configuration of $k\Pi_1$ is $M_0 = (a^3p, p, p, \lambda)$. The only applicable rules are r_1, r_2 and r_3 from C_1 and r_2, r_3 from C_2, C_3. If r_1, r_2 are chosen in C_1 and r_2 in C_2, C_3, then a^3p is rewritten by r_1, r_2 in C_1 and p in C_2, C_3 by r_2; then three a's stay in C_1, three b's go non-deterministically to C_2, C_3, three c's go to compartment labelled 3, C_4, and each p in C_2, C_3 stays in its compartment. Let us assume that two of them go to C_2 and one to C_3. Hence, the next configuration is $M_1 = (a^3p, b^2p, bp, c^3)$. If in the next step the same rules are applied identically in the first compartment, C_1, and rules r_1, r_2 are used in C_2 and r_1, r_3 in C_3, then the next configuration is $M_2 = (a^3p, b^2c^2p, bc, c^6)$. If now r_1, r_3 are used in C_1, with r_1 used in the same way and r_1, r_3 in C_2 (no rule is available in C_3) then $M_3 = (a^3, b^2c^4, b^2c, c^9)$; this is a final configuration as there is no promoter to trigger a further step.

Observation. If $r_1 : a \rightarrow a(b,2)(c,3)\ \{\geq p\}$ from C_1 is changed to $r_1 : a \rightarrow a(b,2)(c,*)\ \{\geq p\}$, then the three resulting c's (obtained after applying r_1 to a^3) go non-deterministically to any of the three neighbours of C_1.

Example 3. Let us reconsider the example above enriched with rules dealing with the system structure. First we will show how the system handles the multiplication of compartments with label 2, C_2 and C_3, when a certain condition holds; we will consider the guard $\geq b^2 \geq p$. In this case the new kP system, denoted $k\Pi_2$, will have the same structure and content as $k\Pi_1$ except R_2^σ which is now defined as follows
R_2^σ is $R_2 = R_2^{(1)} \cup R_2^{(2)}$, where $R_2^{(1)} = \{r_1 : (b/c)\ \{\geq p\}; r_2 : p \rightarrow p; r_3 : p \rightarrow \lambda\}$, $R_2^{(2)} = \{r_4 : []_2 \rightarrow []_2[]_2\ \{\geq b^2 \geq p\}\}$ and $\sigma_2 = Lab(R_2^{(1)})^* Lab(R_2^{(2)})$.
We can notice that the regular expression σ_2 tells us that first the rewriting rules are applied in a maximal parallel manner and then one of system structure rules is chosen to be executed.

If the system follows the same pathway as $k\Pi_1$ then M_2 shows a different configuration given that in C_2 after applying $R_2^{(1)}$ in a maximal parallel manner, $R_2^{(2)}$ is applied as indicated by σ_2, when the guard of r_4 is true. The compartment C_2 is divided into two compartments, $C_{2,1}, C_{2,2}$, with the same label 2 and appearing on positions 2 and 3 in the new configuration, $M_2' = (a^3p, b^2c^2p, b^2c^2p, bc, c^6)$; the new compartments labelled 2 are linked to compartment C_1. In the next step both are divided as they contain the guard triggering the membrane division rule r_4. The process will stop when either p will be rewritten to λ or b^2 stops coming.

If we aim to either dissolve or disconnect a compartment labelled by 2 from compartment C_1, once a certain condition is true, for instance b^2c^2p appears in it, then two more rules will be added to $R_2^{(2)}$, namely
$r_5 : []_2 \rightarrow \lambda\ \{\geq b^2 \geq c^2 \geq p\}$, $r_6 : []_2 - []_1 \rightarrow []_2; []_1\ \{\geq b^2 \geq c^2 \geq p\}$. The same regular expression, $\sigma_2 = Lab(R_2^{(1)})^* Lab(R_2^{(2)})$, is used, but in this case $R_2^{(2)}$ contains three elements and at most one is applied at each step, in every compartment with label 2.

3 Neural-like P Systems and P Systems with Active Membranes versus kP Systems

In order to prove how powerful and expressive kP systems are, we will show how two of the most used variants of P systems are simulated by kP system. More precisely, we will show how neural-like P systems and P systems with active membranes are simulated by some reduced versions of kP systems.

Definition 3. *A* neural-like P system *(tissue P system with states) of degree n is a construct $\Pi = (O, \sigma_1, ...\sigma_n, syn, i_0)$ ([10], p. 249), where:*

- *O is a finite, non-empty set of objects, the* alphabet*;*
- *$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$, $1 \leq i \leq n$, represents a* cell *and*
  - *Q_i is the finite set of* states *of cell σ_i;*
  - *$s_{i,0} \in Q_i$ is the* initial state*;*
  - *$w_{i,0} \in O*$ is the* initial multiset *of objects contained in cell σ_i;*
  - *R_i is a finite set of* rewriting and communication rules*, of the form $sw \rightarrow s'xy_{go}z_{out}$; when such a rule is applied, x will replace w in cell σ_i, the objects from y will be sent to neighbouring cells, according to the transmission mode (see th next Observation) and the objects from z will be sent out into the environment; cell σ_i will move from state s to s';*
- *$syn \subseteq \{1, ..., n\} \times \{1, ..., n\}$, the connections between cells,* synapses*;*
- *i_0 is the output cell.*

Observations.

1. For such systems, three processing modes are considered, called "max", "min", "par", and three transmission modes, namely "one", "repl", "spread". For formal definitions and other details we refer to [10].
2. We denote by *simple neural-like P systems* the class of P systems given by Definition 3, with rewriting and communication rules $sw \rightarrow s'x(a_1, t_1) \cdots (a_p, t_p)$, where t_h, $1 \leq h \leq p$, denotes the target cell (σ_h), and processing mode "max", transmission mode defined by the target indications mentioned in each rule.

Notation. For a given P system, Π, the set of numbers computed by Π will be denoted by $M(\Pi)$.

Theorem 1. *If Π is neural-like P system of degree n, then there is a kP system, Π', of degree n and using only rules of type (a), rewriting and communication rules, simulating Π and such that $M(\Pi') \subseteq M(\Pi) \cup \{2\}$.*

Proof. Let Π be a simple neural-like P systems of degree n, as defined above. We construct Π' as follows:
$\Pi' = (A, L, IO, \mu, C_1, ..., C_n)$ where:

- $A = O \cup (\bigcup_{1 \leq i \leq n} Q_i) \cup \{\gamma\}$; γ is a new symbol neither in O nor in $\bigcup_{1 \leq i \leq n} Q_i$;
- $L = \{1, ..., n\}$; $IO = \emptyset$;
- $\mu = syn$;
- $C_i = (i, w'_{i,0}, R_i'^{\sigma}), 1 \leq i \leq n$; and
  - $w'_{i,0} = \gamma, 1 \leq i \leq n$;
  - R'_i contains the following rules:
    1. $\gamma \rightarrow s_{i,0}tw_{i,0}$, where $s_{i,0}, w_{i,0}$ are the initial state and initial multiset, respectively, associated with cell σ_i, and $t \in Q^{(i)}_{s_{i,0}}$. For $s \in Q_i$, denote by $Q^{(i)}_s = \{t | t \in Q_i, sx \rightarrow ty \in R_i\}$; i.e., $Q^{(i)}_s$ gives, when the cell σ_i is in state s, all the states where σ_i can move to. In the first step, in

compartment C_i, a rule $\gamma \to s_{i,0}tw_{i,0}$ is applied and the current multiset becomes $w'_i = s_{i,0}w_{i,0}$.

2. For each pair $(s,t), t \in Q_s^{(i)}$, there are rules
$sx_i \to ty_i \in R_i, 1 \le i \le p \quad (*)$.
If there are no rules in R_i from s to t then another pair is considered. For the above rules, the following rules are considered in R'_i:
$x_i \to y_i \ \{= s = t\}$, $1 \le i \le p$, and $st \to tq \ \{\ge x_1|...| \ge x_p\}$, $q \in Q_t^{(i)}$ $(**)$
In the above guards the notation $\ge x_i$, if $x_i = a_{i,1} \ldots a_{i,l_i}$, denotes $\ge a_{i,1} \ldots \ge a_{i,l_i}$. The rules $(**)$ make use of guards; the first p rules are applied iff the current multiset contains one s and one t, whereas the last one is applicable iff at least one or more of the occurrences of one of the multisets x_i, $1 \le i \le p$, is included in the current multiset. Clearly, in state s only the rules $(*)$ of Π are applicable for this P system, depending on the availability of the multisets occurring on the left hand side of them; the next state Π is moving to is t. Similarly, in Π' only the rules denoted by $(**)$ are applicable; the rule $st \to tq \ \{\ge x_1|...| \ge x_p\}$ is applied once whereas the first p rules are applied as many times as their corresponding $(*)$ rules are applied.

If the set $Q_t^{(i)}$ used in $st \to tq \ \{\ge x_1|...| \ge x_p\}$ of $(**)$ is empty, i.e., there are no rules from state t, then the rule is replaced by $st \to \lambda$. When $Q_{s_{i,0}}^{(i)} = \emptyset$ then the rule $\gamma \to w_{i,0}$ is introduced in R'_i.

At any moment the component C_i of the kP system Π' contains a multiset which is the multiset of σ_i augmented by the current state of σ_i, s, and one of the next states, t, if it exists.

The process will stop in component C_i of Π' when no pair of rules of type $(**)$ is applicable, which means no $sx_i \to ty_i$ rule is applicable in state s.

The multiset $M(\Pi')$ contains $M(\Pi)$ and maybe two states s, t occurring in the last step of the computation. Hence $M(\Pi') \subseteq M(\Pi) \cup \{2\}$. □

Comments.

1. The above simulation can be assessed with respect to number of compartments, objects and rules as well as the computation steps.
2. When rules $sw \to txy_{go}$ are used in the "spread" mode, this means that any $a \in O$ occurring in y may go to any of the neighbours. In this case if $y = y_1ay_2$ then for each such $a \in O$, the rules of R'_i corresponding to $sw \to txy_{go}$, denoted $(**)$ above, will show $w \to xy \ \{= s = t\}$ replaced by $w \to xy_1(a,j)y_2 \ \{= s = t\}$, where j the label of one of the neighbours of the current compartment. For "one" mode all a's in y will point to the same target, j, for all neighbours of the compartment i.
3. The transmission replicative mode - when a symbol is sent to all the neighbours, can also be simulated. Indeed if $j_1, \cdots, j_h$ are the neighbours of i, then $w \to xy_1(a,j)y_2$ is transformed into $w \to xy_1(a,j_1)\cdots(a,j_h)y_2$ for each a.

4. If a rewriting rule contains z_{out} on its right side, i.e., $sw \to txy_{go}x_{out}$ then in the set of rules transcribing it, $w \to \alpha$, we will have $\alpha = xy_1(a, j)y_2z'$, where if $z = a_1 \cdots a_k$, then $z' = a'_1 \cdots a'_k$; also rules (a'/λ) will be added to R'_i, for any $a \in O$. In this way in the next step all the prime elements are sent out into the environment. If there is a need to synchronize the behaviour of the system with the environment then this should be done a bit differently. For this transmission mode the kP system is a bit more complex as it must contain input-output rules and the environment definition needs to be considered.
5. If we want to simulate the "min" processing mode then this can be obtained by specifying the sequential behaviour of the component i - by changing the regular expression of the component.

We study now how P systems with active membranes are simulated by kP systems. In this case we are dealing with a cell-like system, so the underlying struture is a tree and a set of labels (types) for the compartments of the system. The system will start with a number of compartment and its structure will evolve. In the study below it will be assumed that the number of compartments simultaneously present in the system is bounded.

Definition 4. *A P system with active membranes of initial degree n is a tuple (see [11], Chapter 11)* $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n,0}, R, i_0)$ *where:*

- O, $w_{1,0}, \ldots, w_{n,0}$ *and* i_0 *are as in Definition 3;*
- H *is the set of labels for compartments;*
- μ *defines the tree structure associated with the system;*
- R *consists of rules of the following types*
  - *(a)* rewriting rules*:* $[u \to v]_h^e$*, for* $h \in H$*,* $e \in \{+, -, 0\}$ *(set of electrical charges),* $u \in O^+$*,* $v \in O^*$*;*
  - *(b)* in communication rules*:* $u[]_h^{e_1} \to [v]_h^{e_2}$*, for* $h \in H$*,* $e_1, e_2 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(c)* out communication rules*:* $[u]_h^{e_1} \to []_h^{e_2}v$*, for* $h \in H$*,* $e_1, e_2 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(d)* dissolution rules*:* $[u]_h^e \to v$*, for* $h \in H \backslash \{s\}$*, s denotes the skin membrane (the outmost one),* $e \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(e)* division rules for elementary membranes*:* $[u]_h^{e_1} \to [v]_h^{e_2}[w]_h^{e_3}$*, for* $h \in H$*,* $e_1, e_2, e_3 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v, w \in O^*$*;*

The following result shows how a P system with active membranes starting with n_1 compartments and having no more than n_2 simultaneously present ones can be simulated by a kP systems using only rules of type (a).

Theorem 2. *If* Π *is a P system with active membrane having* n_1 *initial compartments and utilising no more than* n_2 *compartments at any time, then there is a kP system,* Π'*, of degree* 1 *and using only rules of type (a), rewriting and communication rules, such that* Π' *simulates* Π*.*

Proof. Let us denote $J_0 = \{(i,h) | 1 \leq i \leq n_2, h \in H\}$; for a multiset $w = a_1 \ldots a_m$, (w,i,h), $(i,h) \in J_0$, denotes $(a_1,i,h) \ldots (a_m,i,h)$. Let us consider the P system with active membranes, $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n_1,0}, R, i_0)$. The polarizations of the n_1 compartments are all 0, i.e., $e_1 = \ldots = e_{n_1} = 0$.

We construct Π' as follows:
$\Pi' = (A, L, IO, \mu', C_1)$ where:

- $A = \bigcup_{(i,h) \in J_0} \{(a,i,h) | a \in O \cup \{+,-,0\} \cup \{\delta\}\}$, where δ is a new symbol; let us denote by $= \overline{\delta_{all}}$ the guard $= \overline{(\delta,1,1)} \ldots = \overline{(\delta, n_2, |H|)}$, $|H|$ is the number of elements in H ($= \overline{\delta_{all}}$ stands for none of the (δ,i,h), $(i,h) \in J_0$);
- $L = \{1\}$; $IO = \emptyset$; $\mu' = []_1$;
- $C_1 = (1, w'_{1,0}, R_1'^{\sigma})$, and
  - $w'_{1,0} = (w_{1,0}, 1, h_1) \ldots (w_{n_1,0}, n_1, h_{n_1})(e_1, 1, h_1) \ldots (e_{n_1}, n_1, h_{n_1})$, $e_1 = \ldots = e_{n_1} = 0$; let $J_c = J_0 \setminus \{(i,h_i) | 1 \leq i \leq n_1\}$ (J_c denotes indexes available for new compartments and $J_0 \setminus J_c$ the set of indexes of the current compartments);
  - R'_1 contains the following rules
    - (a') for each $h \in H$ and each rule $[u \to v]_h^e \in R$, $e \in \{+,-,0\}$, we add the rules $(u,i,h) \to (v,i,h)$ $\{= (e,i,h) = \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$; these rules are applied to every multiset containing elements with $h \in H$, only when the polarization (e,i,h) appears and none of the (δ, j, h') appears;
    - (b') for each $h \in H$ and each rule $u[]_h^{e_1} \to [v]_h^{e_2} \in R$, $e_1, e_2 \in \{+,-,0\}$, we add the rules $(u,j,l)(e_1,i,h) \to (v,i,h)(e_2,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$, j is the parent of i of label l; these rules will transform (u,j,l) corresponding to u from the parent compartment j to (v,i,h) corresponding to v from compartment i of label h, the polarization is changed; for each polarization, (e_1,i,h) only one single rule can be applied at any moment of the computation;
    - (c') for each $h \in H$ and each rule $[u]_h^{e_1} \to []_h^{e_2} v \in R$, $e_1, e_2 \in \{+,-,0\}$, we add the rules $(u,i,h)(e_1,i,h) \to (v,j,l)(e_2,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$, j is the parent of i of label l;
    - (d') for each $h \in H$ and each rule $[u]_h^e \to v \in R$, $e \in \{+,-,0\}$, we add the rules $(u,i,h)(e,i,h) \to (v,j,l)(\delta,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$, j is the parent of i of label l; all the elements corresponding to those in compartment i must be moved to j - this will happen in the presence of (δ,i,h) when no other transformation will take place; this is obtained by using rules $(a,i,h) \to (a,j,l)$ $\{= (\delta,i,h)\}$, $a \in O$ and $(\delta,i,h) \to \lambda$ $\{= (\delta,i,h)\}$; the set of available indexes will change now to $J_c = J_c \cup \{(i,h)\}$;
    - (e') for each $h \in H$ and each rule $[u]_h^{e_1} \to [v]_h^{e_2}[w]_h^{e_3} \in R$, $e_1, e_2, e_3 \in \{+,-,0\}$; if j_1, j_2 are the indexes of the new compartments, we add $(u,i,h)(e_1,i,h) \to (v,j_1,k_1)(e_2,j_1,k_1)(w,j_2,k_2)(e_3,j_2,k_2)(\delta,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$; the content corresponding to compartment i should be moved to j_1 and j_2, hence rules $(a,i,h) \to (a,j_1,k_1)(a,j_2,k_2$ $\{= (\delta,i,h)\}$, $a \in O$ and finally $(\delta,i,h) \to \lambda$ $\{= (\delta,i,h)\}$; J_c is updated, $J_c = J_c \cup \{(i,h)\} \setminus \{(j_1,k_1),(j_2,k_2)\}$.

The size of the multiset obtained in i_0 by using Π computation is the same as the size of the multiset in Π, when only (a, i_0, h) are considered, minus 1 (the polarization is also included). □

4 Case Study - Static Sorting

In this section we analyze the newly introduced kP systems by comparing them with established P system classes by using them to specify a static sorting algorithm. This algorithm was first written with symport/antiport rules [4] and then reconsidered in some other cases [2]. The specification below mimics this algorithm.

4.1 Static Sorting with kP Systems

Let us consider a kP system having the following $n = 6$ compartments:
$C_i = (i, w_{i,0}, R_i^{\sigma})$, $1 \leq i \leq n$, where
$w_{1,0} = a^3; w_{2,0} = a^6 p; w_{3,0} = a^9; w_{4,0} = a^5 p; w_{5,0} = a^7; w_{6,0} = a^8 p.$

The rules in compartment R_i, $1 \leq i \leq n$, are :
$r_1 : a \rightarrow (b, i-1) \; \{\geq p\}$, only for $i > 1$
$r_2 : p \rightarrow p'$
$r_3 : p' \rightarrow (p, i-1)$, for i even and $r'_3 : p' \rightarrow (p, i+1)$, for i odd
$r_4 : ab \rightarrow a(a, i+1)$, $i < n$
$r_5 : b \rightarrow a$, $i < n$.

We assume that any two compartments, C_i, C_{i+1}, $1 \leq i < n$, are connected. The aim of this problem is to order the content of these compartments such that the highest element (a^9) will be in the left most compartment, C_1, and the smallest one (a^3) in the right most compartment, C_n, ($n = 6$).

Remarks:

- the set of objects is $A = \{a, b, p, p'\}$;
- compartment C_i has the label i, $1 \leq i \leq n$; so any two compartments have distinct labels;
- the rule r_1 is absent from the compartment C_1;
- the last two rules, r_4, r_5, are only present in compartments C_1 to C_{n-1};
- for $n = 2k + 1$ we need an auxiliary compartment, C_{n+1}, which will start with an initial multiset p and will contain a set of rules with $r_2 : p \rightarrow p'$ and $r_3 : p' \rightarrow (p, n)$; whereas C_n should have an additional rule $r'_3 : p' \rightarrow (p, n+1)$;
- the regular expression corresponding to the execution of the rules in a compartment C_i is $\sigma_i = \{r_1, r_2, r_3, r_4\}^* \{r_5\}^*$, if i is even; for odd values of i, r_3 is replaced by r'_3; the regular expression tells us that firstly the rules from the first set are applied in a maximal parallel manner and then r_5, also in a maximal way.

Observation. The regular expression, σ_i, describes an order relationship, $r_1, r_2, r_3, r_4 > r_5$. So we can replace this kP system by a P system with promoters and having an order relationships on the set of rules associated with each membrane.

The table below presents the first steps of the computation. In the first step the only applicable rules are r_1, r_2; given the presence of the promoter p, rule r_1 moves all a's from each even compartment to the left compartment as b's and rule r_2 transforms the promoter into p'. Next, rules r_3, r_4, r_5 are applicable; first r_3 and r_4 are applied, this means p' is moved as p to the left compartments and for each ab an a is kept in the current compartment and a b is moved as an a to the right compartment; finally, the remaining b's, if any, are transformed into a's. These two steps implement a sort of comparators between two adjacent compartments moving to the left bigger elements. In the previous steps the comparators have been considered between odd and even compartments. In the next step the promoters appear in even compartments and the comparators are now acting between an even and an odd compartment. The algorithm does not have a stopping condition. It must stop when no changes appear in two consecutive steps. Given that the algorithm must stop in maximum $2(n-1)$ steps, then we can introduce such a counter, c, in each compartment and rules $c \to c_1$, $c_i \to c_{i+1}$, $1 \leq i \leq 2(n-1)-2$ and $c_{2(n-1)-1}p \to \lambda$. These rules should be executed before the rest, so the regular expression associated with them should be a prefix of the regular set associated with each compartment.

Compartments - Step	C_1	C_2	C_3	C_4	C_5	C_6
0	a^3	a^6p	a^9	a^5p	a^7	a^8p
1	a^3b^6	p'	a^9b^5	p'	a^7b^8	p'
2	a^6p	a^3	a^9p	a^5	a^8p	a^7
3	a^6p'	a^3b^9	p'	a^5b^8	p'	a^7
4	a^6	a^9p	a^3	a^8p	a^5	a^7p
5	a^6b^9	p'	a^3b^8	p'	a^5b^7	p'

Observation. Bounded number of labels! The above solution is using n labels for n compartments. As the rules are the same in each compartment, with two exceptions involving the components at both ends of the system (compartments C_1 and C_n), it is natural to look for a solutions with a bounded number of labels. If we use the same label everywhere except for the two margins then we face the problem of replacing the rules using targets with different rules where the targets are now the new labels; if these are the same we can no longer distinguish between left and right neighbours, so we should have at least two distinct ones. Additionally, we have to distinguish odd and even positions. Consequently, four labels, and two more for the two ends are enough. Are there further simplifications? The answer to this question and the solution in this case are left as exercises to the reader.

4.2 Static Sorting with States

We consider the same n-compartment tissue-like P system structure as in the previous subsection. Additionally, in this case, the rules in each compartment use states; an order relationship between rules in each compartment is also considered. Initial states are s_1 in odd compartments and s_0 otherwise; the content of the 6 regions is illustrated by the first line, step 0, of the table below.

The addition of states is potentially very useful from a modelling point of view since many widely-used modelling languages are state-based and, therefore, such rules were a strong candidate for inclusion in our kP system model. However, as shown below, states can be effectively simulated by rewriting rules, as shown below .

For the algorithm considered, the rules in each compartment and the order relationships are as follows

Compartment 1:
$r_1 : s_0x \rightarrow s_0y$
$r_2 : s_0y \rightarrow s_1x$
$r_3 : s_1ab \rightarrow s_0a(a, 2)$
$r_4 : s_1b \rightarrow s_0a$
The rules satisfy: $r_1, r_2, r_3 > r_4$.

Compartment i, $2 \leq i \leq n-1$:
$r_1 : s_0a \rightarrow s_1(b, i-1)$
$r_3 : s_1ab \rightarrow s_0a(a, i+1)$
$r_4 : s_1b \rightarrow s_0a$
The rules satisfy: $r_1, r_3 > r_4$.

Compartment n:
$r_1 : s^0a \rightarrow s^1b_{n-1}$
$r_2 : s^1x \rightarrow s^1y$
$r_3 : s^1y \rightarrow s^1z$
$r_4 : s^1z \rightarrow s^0x$

Membranes - Step	C_1	C_2	C_3	C_4	C_5	C_6
0	$s^1 : a^3x$	$s^0 : a^6$	$s^1 : a^9$	$s^0 : a^5$	$s^1 : a^7$	$s^0 : a^8x$
1	$s^1 : a^3b^6x$	$s^1 : \_$	$s^1 : a^9b^5$	$s^1 : \_$	$s^1 : a^7b^8$	$s^1 : x$
2	$s^0 : a^6x$	$s^1 : a^3$	$s^0 : a^9$	$s^1 : a^5$	$s^0 : a^8$	$s^1 : a^7y$
3	$s^0 : a^6y$	$s^1 : a^3b^9$	$s^1 : \_$	$s^1 : a^5b^8$	$s^1 : \_$	$s^1 : a^7z$
4	$s^1 : a^6x$	$s^0 : a^9$	$s^1 : a^3$	$s^0 : a^8$	$s^1 : a^5$	$s^0 : a^7x$
5	$s^1 : a^6b^9$	$s^1 : \_$	$s^1 : a^3b^8$	$s^1 : \_$	$s^1 : a^5b^7$	$s^1 : x$

In the case where we have an odd number of compartments, the $n-$th region must contain an y instead of x. Thus the starting configuration for $n = 7$ is the

following:
$w_{1,0} = a^3x; w_{2,0} = a^6; w_{3,0} = a^9; w_{4,0} = a^5; w_{5,0} = a^7; w_{6,0} = a^8; w_{7,0} = a^{13}y.$

4.3 Static Sorting with P Systems using Polarizations on Membranes

We now use cell-like P systems with active membranes to specify the same algorithm. P systems with active membranes were introduced with the primary aim of solving NP-complete problems in polynomial (often linear) time [11]. The key features of this variant is the possibility of multiplying the number of compartments during the computation process by using membrane division rules in addition to multiset rewriting and communication rules. Each membrane can have one of the three electrical charges $\{+, -, 0\}$ and a rule can only be executed if the membrane has the required electrical charge; a rule can also change the polarization of the membrane when objects cross it (either in or out).

In our static sorting example compartments with two states were used, so, when the algorithm is implemented using electrical charges, it is expected that two electrical charges would suffice. Indeed, from the list of rules below one may observe that 0 and + are the only polarizations utilised.

There is, however, a problem with this approach, arising from the rule application strategy. In P systems with membrane division and polarizations, only one rule which can change the polarization of a membrane can be applied per step [7]. The sorting algorithm however, employs maximal parallel communication rules to operate the *comparator* procedure between membranes. In order to correctly implement this procedure we will accept maximal parallel communication rules which change the charge of the membrane they traverse to/from *if and only if* they target the same final polarization.

In the case of P systems with polarizations on membranes we will use a cell-like structure with $n = 6$ regions defined below with the initial multisets included and initial polarizations; the implementation of the static sorting with P systems with polarization on membranes is using priorities over the sets of rules.

$$\mu = [[[[[[[a^3x_1]^0_1 a^6x_1]^+_2 a^9x_1]^0_3 a^5x_1]^+_4 a^7x_1]^0_5 a^8x_1]^+_6]^0_{aux}$$

Rules:
"Comparator" rules:
$r_1 : a[]^0_j \rightarrow [b]^0_j, 1 \leq j \leq n;$
$r_2 : [ab]^0_j \rightarrow a[a]^+_j, 1 \leq j \leq n;$
$r_3 : [b \rightarrow a]^0_j, 1 \leq j \leq n;$

Rules for switching polarities between adjacent membranes:
$r_4 : [x_1 \rightarrow x_2]^i_j, 1 \leq j \leq n;$
$r_5 : [x_2]^0_j \rightarrow y_1[]^+_j, 1 \leq j \leq n;$
$r_6 : [x_2]^+_j \rightarrow y_1[]^0_j, 1 \leq j \leq n;$
$r_7 : [y_1 \rightarrow y_2]^i_j, 1 < j \leq n + 1;$

$r_8 : y_2[]_j^0 \rightarrow [x_1]_j^+, 1 \leq j \leq n;$
$r_9 : y_2[]_j^+ \rightarrow [x_1]_j^0, 1 \leq j \leq n;$
where $i \in \{0, +\}$; and the order relationship $r_1, r_2, r_4, r_5, r_6, r_7, r_8, r_9 > r_3$.

M/S	$[]_1$	$[]_2$	$[]_3$	$[]_4$	$[]_5$	$[]_6$	$[]_{aux}$
0	$[a^3x_1]^0$	$[a^6x_1]^+$	$[a_9x_1]^0$	$[a^5x_1]^+$	$[a^7x_1]^0$	$[a^8x_1]^+$	$[\_]^0$
1	$[a^3b^6x_2]^0$	$[x_2]^+$	$[a^9b^5x_2]^0$	$[x_2]^+$	$[a^7b^8x_2]^0$	$[x_2]^+$	$[\_]^0$
2	$[a^6]^+$	$[a^3y_1]^0$	$[a^9y_1]^+$	$[a^5y_1]^0$	$[a^8y_1]^+$	$[a^7y_1]^0$	$[y_1]^0$
3	$[a^6]^+$	$[a^3b^9y_2]^0$	$[y_2]^+$	$[a^5b^8y_2]^0$	$[y_2]^+$	$[a^7y_2]^0$	$[y_2]^0$
4	$[a^6x_1]^0$	$[a^9x_1]^+$	$[a_3x_1]^0$	$[a^8x_1]^+$	$[a^5x_1]^0$	$[a^7x_1]^+$	$[\_]^0$
5	$[a^6b^9x_2]^0$	$[x_2]^+$	$[a_3b^8x_2]^0$	$[x_2]^+$	$[a^5b^7x_2]^0$	$[x_2]^+$	$[\_]^0$

There are no additional requirements in the case where $n = 2k + 1$, however we always entail an extra auxiliary membrane to enable *out* communication of the $n-$th membrane, therefore allowing it to switch polarity.

A similar implementation of the static sorting algorithm can be obtained by using P systems with labels on membranes. As illustrated in [1], we can encode electrical charges in strings of the membrane labels, in order to differentiate between the two necessary states. For each membrane h_i we synthesise its complementary label h'_i, which is changed to by a communication rule. We leave this as an exercise to the reader.

A number of (preliminary) conclusions can be drawn from the above case study:

- kP systems are conceptually closer to tissue P systems than cell-like P systems; in our case studies, this is reflected by the similarity between the specifications using kP systems and tissue P systems, respectively. On the other hand, the model realized using the cell-like P system variant is significantly more complex.
- In terms of complexity, the three implementations are roughly equivalent. The kP system executes in each step one more rule then the P system with states; this rule is either r_2 or r_3 (dealing with p). On the other hand, the number of rules applied in each compartment for every step by cell-like P systems is similar to the case of kP systems.

5 Conclusions

The kP system introduced in this work represents a low level specification language. Its syntax and informal semantics and some examples have been introduced and discussed. A case study based around a simple sorting algorithm has allowed us to compare different specifications of this using various types of P systems. In the next stage formal semantics will be defined and an implementation using model checkers (SPIN, Maude) is also expected. Several extensions can be considered for kP systems that may lead to a more flexible and higher level specification language. A first set of extensions refer to ways of defining objects, rules and compartments using indexes over some specified domains. Modules can also be introduced using a syntactic approach, rather than considering additional semantic features [8]. In order to prove the expressive power of kP systems, a more systematic study of simulating important classes of P systems with kP systems will be produced in a forthcoming paper.

Acknowledgement. The work of MG and FI was partially supported by project MuVet, Romanian National Authority for Scientific Research (CNCS – UEFISCDI) grant number PN-II-ID-PCE-2011-3-0688.

References

1. A. Alhazov, L. Pan, Gh. Păun, Trading Polarizations for Labels in P Systems with Active Membranes, *Acta Informatica*, 41, 111-144, 2004.
2. A. Alhazov, D. Sburlan, Static Sorting P Systems. In [5], 215 – 252, 2006.
3. M. Ben-Ari, *Principles of the SPIN Model Checker*, Springer, 2008.
4. R. Ceterchi, C. Martín-Vide, P Systems with Communication for Static Sorting. In *Pre-Proceedings of Brainstorming Week on Membrane Computing, Tarragona, February 2003*, M. Cavaliere, C. Martín-Vide, Gh. Păun, eds., Technical Report no 26, Rovira i Virgili Univ., Tarragona, 101–117, 2003.
5. G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez, eds., *Applications of Membrane Computing*, Springer, 2006.
6. M. Clavel, F.J. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, Maude: Specification and Programming in Rewriting Logic, *Theoretical Computer Science*, 285, 187 – 243, 2002.
7. D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A Uniform Family of Tissue P Systems with Cell Division Solving 3-COL in a Linear Time, *Theoretical Computer Science*, 404, 76 – 87, 2008.
8. M. Gheorghe, V. Manca, F.J. Romero-Campero, Deterministic and Stochastic P Ssytems for Modelling Cellular Processes, *Natural Computing*, 9, 457–473, 2010.
9. R. Nicolescu, M. J. Dinneen, Y.-B. Kim, Structured Modelling with Hyperdag P Systems: Part A. In *Membrane Computing, Seventh Brainstorming Week, BWMC 2009, Sevilla, Spain, February 2009*, R. Gutiérrez-Escudero, M. A Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, eds., Universidad de Sevilla, 85 – 107, 2009.
10. Gh. Păun, *Membrane Computing: An Introduction*, Springer, 2002.
11. Gh. Păun, G. Rozenberg, A. Salomaa, eds., *Handbook of Membrane Computing*, Oxford University Press, 2010.

Frontiers of Membrane Computing: Open Problems and Research Topics

Marian Gheorghe[1], **Gheorghe Păun**[2,3], **Mario J. Pérez-Jiménez**[3] – **Editors**

[1] Department of Computer Science, University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
m.gheorghe@dcs.shef.ac.uk

[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
gpaun@us.es, marper@us.es

Summary. This is a list of open problems and research topics collected after the Twelfth Conference on Membrane Computing, CMC 2012 (Fontainebleau, France (23 - 26 August 2011), meant initially to be a working material for Tenth Brainstorming Week on Membrane Computing, Sevilla, Spain (January 30 - February 3, 2012). The result was circulated in several versions before the brainstorming and then modified according to the discussions held in Sevilla and according to the progresses made during the meeting. In the present form, the list gives an image about key research directions currently active in membrane computing.

Introduction

The idea of compiling a collection of open problems and research topics in membrane computing (MC) occurred during the Twelfth International Conference on Membrane Computing, CMC 12, held in Fontainebleau, Paris, France, from 23 to 26 of August, 2011 (see http://cmc12.lacl.fr/). The invitation to contribute to such a collection was formulated during CMC 12 (and after that reinforced by email) and several researchers answered this call. The result was circulated under the name of "mega-paper" (*mega* because it has much more co-authors than any other paper in MC...), meant to be a working material for the Tenth Brainstorming Week on Membrane Computing, Sevilla, Spain, January 30 - February 3, 2012 (BWMC 10). During CMC 12 there were also discussions and suggestions regarding some other topics which are not developed here; for this reason we briefly mention (some of) them: exploring more systematically hypercomputation research ideas within MC area; focussing on translations between different classes of membrane systems (P systems) and studying complexity aspects related to these translations (the goal being to import results from a branch of MC to another one); identifying the most "natural" applications of P systems in modeling biological processes and producing a set of coherent and convincing case studies (a research volume on such applications in biology is now in progress); investigate in more details the dP automata, their efficiency and connections with communication complexity; look for biological applications of spiking neural P systems.

Before presenting an overview of the paper we mention [7], [8] as key references for general MC topics. More specific MC topics, like MC and process calculi [1], interplay between MC and DNA computing [6] and conformon MC systems [3], are also well-established. Applications of MC in various areas can be found in [2].

The initial "mega-paper" was changed several times, incorporating discussions and progresses carried out during BWMC 10. The present version is considered a "closed" one (although such a project can never be closed); for further results related to the problems collected here the reader is invited to follow the MC website from [9]. In particular, one can find there the proceedings volumes, with all papers emerged in connection with the brainstorming.

The texts received from the contributors were revised by their authors after BWMC 10, and appear below in the final form they have been submitted, with minimal editorial changes. In most cases, one gives the necessary (minimal) definitions, as well as the relevant bibliography. Of course, the reader is supposed to be familiar with basic elements of MC – for instance, from the sources mentioned at the end of this introduction. A quick introduction to MC is given at the beginning of this paper, just to help the reader not familiar with this research area to have a flavor of it. At the beginning of each section there are mentioned the main notions, from MC and from computability in general, supposed to be known in order to understand the problems which follow (sometimes, part of these notions are briefly introduced together with the problems). The authors of each "section" are mentioned, with affiliations and email addresses, so that the interested reader can contact them for further details, clarifications and cooperation in solving the problems.

The order in which the problems are given below goes, approximately, from general issues to theory and then to applications. In what concerns the computability topics, there are sections devoted to both power and efficiency of P systems, considering them as numbers or strings generators or acceptors, in "old" versions (symport/antiport, catalytic, spiking neural P systems) or in recently introduced forms (polymorphic, dP systems), looking for generalizations (e.g., for "kernel P systems") or for classic notions of language theory not yet extended to MC (such as control words); computational complexity is a vivid direction of investigation, addressing both time and space complexity (defining specific complexity classes, comparing them with existing classes, looking for possibilities of solving computationally hard problems (typically, **NP**-complete problems) in a polynomial time, by making use of the massive parallelism of P systems and trading-off space for time, with the space obtained by means of biologically inspired operations, such as membrane division and membrane creation). The term "fypercomputing" (following the model of "hypercomputing" = "passing beyond the Turing barrier", with the initial "f" coming from "fast") tries to call attention to a systematic study of "passing polynomially beyond the **NP** barrier". All classes of P systems are considered: cell-like, tissue-like, (spiking) neural, and numerical. Moving to applications, one mentions issues related to the semantics, formal verification, possible bridges with reaction systems (a younger "sister" research area of natural computing, inspired from biochemistry). The applications refer both to the simulation of biological and bio-medical processes and to (somewhat unexpected) applications in approximate optimization (basically, distributed evolutionary algorithms, with the distribution controlled by means of membranes, and borrowing ingredients from MC), robotics (mobile robots controlled by means of numerical P systems), and computer graphics, as well as to more speculative ideas, dealing, for instance, with the functioning of the brain.

The nature of questions range from local/technical open problems, asking to improve existing results, especially for a better delimitation of the borderline between universality and non-universality, between efficiency and non-efficiency (in particular, concerning the influence of some qualitative parameters, such as the number of membranes, the size of the rules, or qualitative features, such as the difference between deterministic and non-deterministic systems, using or not various types of rules), to "strategic" issues, for instance, relating MC with other research areas, such as computer science, biology, ecology, robotics and so on. Of course, many other precise problems or research ideas circulate within the MC community (or can be found in recent papers; see also the previous brainstorming volumes, where many problems are formulated, sometimes given in explicit lists; the "fate" of some of these open problems is recalled in the paper Gh. Păun, "Tracing Some Open Problems in Membrane Computing", *Romanian J. of Information Science and Technology*, 10, 4 (2007), 303–314). Similarly, some of the problems proposed in the present paper or variants of them were already circulated within the MC community also before, a fact which should call attention to them (as an indication of both interest and difficulty).

We are aware, on the one hand, that many other authors, who have not answered our request (in time), would have other problems to propose, and, on the other hand, that many people keep for them, for their immediate research, the "juicy" topics... Anyway, we hope that this collection will both raise the interest of the reader in approaching MC and, maybe, in participating in the future editions of the yearly BWMC.

Because several sections below refer to the CMC 12 pre-proceedings and proceedings volumes, we also mention them below – [4] and [5], respectively.

References

1. G. Ciobanu: *Membrane Computing. Biologically Inspired Process Calculi.* The Publishing House of the "Al.I. Cuza" University, Iaşi, 2010.
2. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing.* Springer, Berlin, 2006.
3. P. Frisco: *Computing with Cells. Advances in Membrane Computing.* Oxford Univ. Press, 2009.
4. M. Gheorghe, Gh. Păun, S. Verlan, eds.: *Twelfth International Conference on Membrane Computing, CMC12, Fontainebleau, France, 23–26 August 2011.* LACL, Univ. Paris Est – Créteil Val de Marne, 2011.
5. M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, S. Verlan, eds.: *Membrane Computing. 12th International Conference, CMC 2011, Fontainebleau, France, August 2011, Revised Selected Papers*, LNCS 7184, Springer, Berin, 2012.
6. A. Păun: *Computability of the DNA and Cells. Splicing and Membrane Computing.* SBEB Publishing, Choudrant, Louisiana, USA, 2008.
7. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002 (Chinese translation in 2012).
8. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing.* Oxford Univ. Press, 2010.
9. The P Systems Website: `www.ppage.psystems.eu`.

Contents

17. Kernel P Systems (M. Gheorghe)
18. Bridging P and R (Gh. Păun)
19. P Systems and Evolutionary Computing Interactions (G. Zhang)
20. Metabolic P Systems (V. Manca)
21. Unraveling Oscillating Structures by Means of P Systems (T. Hinze)
22. Simulating Cells Using P Systems (A. Păun)
23. P Systems for Computational Systems and Synthetic Biology (M. Gheorghe, V. Manca, F.J. Romero-Campero)
24. Biologically Plausible Applications of Spiking Neural P Systems for an Explanation of Brain Cognitive Functions (A. Obtulowicz)
25. Computer Vision (D. Díaz-Pernil, M.A. Gutiérrez-Naranjo)
26. Open Problems on Simulation of Membrane Computing Models (M. García-Quismondo, L.F. Macías-Ramos. M.A. Martínez-del-Amor, I. Pérez-Hurtado, L. Valencia-Cabrera)

1 A Glimpse to Membrane Computing

Membrane computing (MC) is a branch of natural computing (introduced in [1], with the report version of the paper circulated as Turku Center for Computer Science – TUCS Report 208, in November 1998, see `www.tucs.fi`) which aims to abstract computing models from the structure and the functioning of the living cell and from populations of cells (e.g., tissues, organs), including the brain. One of the basic notions is that of a *membrane*, understood as a 3D vesicle, separating "an inside" and "an outside", where *objects* can be placed and where specific biochemistries take place. The membranes can be arranged in a hierarchical structure (like in a cell, hence described by a tree) or in an arbitrary structure (like in tissues, hence described by a graph). The space between a membrane and the membranes placed immediately inside it (parent-children, in a tree) is called *region* or *compartment.* A membrane without any membrane inside is said to be *elementary.* In the case of a cell-like arrangement of membranes, the external membrane is called the *skin.* The space outside the skin membrane is called the *environment* (and similarly is called the space external to all membranes of a tissue-like membrane structure). A membrane structure can be formally represented by a rooted labeled tree (each membrane is identified by a label, which is then associated with the node of the tree associated with the membrane), or, correspondingly, by an expression of labeled parentheses, with a unique external pair of parentheses, corresponding to the skin membrane.

The objects are present in the regions of a membrane structure and in the environment in the form of *multisets*, sets with their elements present in a given number of copies (sets with multiplicities of elements). The multiplicity can be finite (expressed by a natural number) or infinite/arbitrary (we say that an object with this property is of ω multiplicity). For the beginning, let us have in mind only atomic objects, represented by symbols of a given (finite) alphabet, and let us imagine that they correspond to the chemical compounds, from ions to macromolecules, which swim in water in the cell compartments. In this framework, it is convenient to represent the multisets by strings of symbols, with the number of occurrences of a symbol in a string corresponding to the multiplicity of that object in the multiset (that is, any permutation of a string represents the same multiset). These objects react, according to given *evolution rules*. The basic ones (often simply called "evolution rules") are the multiset rewriting rules corresponding to the biochemical reactions taking place in a cell. They are of the form $u \to v$, where u and v are multisets. Many other types of evolution rules are inspired by other biological operations. We mention here only the basic ones: *symport/antiport* correspond to the coupled passage of chemicals through (the protein channels embedded in) the cell membranes, *membrane division* corresponds to mitosis, *membrane creation* and *membrane dissolution* can also be associated with biological processes (the same with exo- and endocytosis, but we do not enter into details). There also are more complex types of rules, or rules inspired from computer science (broadcasting, communication between two membranes placed in a common environment), rules mimicking the way the neurons communicate by means of *spikes* (electrical impulses of identical shapes). Important is that both the rules and the objects are placed in compartments and that the rules act locally, on the objects in the same compartment. Objects can also pass through membranes, both in the cell-like case and in the tissue-like case, hence the compartments cooperate.

There are several ways the rules are applied (several *semantics*). The most investigated one, corresponding to the parallelism of reactions in a solution, is the *maximal parallelism*: a maximal multiset of rules is used, where maximality is defined in the sense of multiset inclusion (no rules can be added to the multiset so that the obtained multiset of rules is still applicable to the multiset of objects present in the respective compartment). When several (maximal) multisets can be applied, the one to use is chosen nondeterministically. Many other possibilities were considered: sequential, limited parallelism, minimal parallelism (the idea is that each compartment which can use a rule – hence it is "alive" – has to use at least one rule, with natural extensions to P systems whose rules are not associated with compartments – as it is the case of symport/antiport systems, where the rules are associated with the membranes). In all these cases, the system is synchronized, a universal clock exists which measures the time in the same way for all membranes and with rules used, synchronously, in each time unit. The natural counterpart is that of asynchronous systems.

Such a device, consisting of membranes, objects, evolution rules, is called a *membrane system* – currently called also a *P system.*

Starting from an *initial configuration* (membranes and objects) of a P system and using the rules according to a chosen strategy, one obtains *computations*, sequences of *transitions* among configurations. If a configuration is obtained such that no rule can be applied, we say that the system *halts.* Several *results* can be associated with a halting computation, for instance, in the form of the number of objects present in the halting configuration in a designated elementary membrane. A P system can then be seen as a generative device, generating a set of numbers: because of nondeterminism, we have several computations, hence several numbers.

Formally, a P system of the basic form (cell-like, with symbol objects, evolving by multiset rewriting rules) can be given as follows (for an alphabet A, we denote by A^* the set of all strings over A, including the empty string λ; $A^* - \{\lambda\}$ is denoted by A^+):

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

m is the *degree* of the system,

O is the alphabet of objects,

μ is the membrane structure, with m membranes,

$w_1, \ldots, w_m \in O^*$ are multisets associated with the m regions of μ,

$R_1, \ldots, R_m$ are finite rules of the form $u \to v$ where u and v are multisets over O with the objects in v also having *target indications* of the form *in, out, here*; an object with indication *out* exits the membrane, one with the indication *here* remains in the same region, and one with the target *in* enters any of the membranes delimiting the region from below, nondeterministically choosing the destination,

i_0 is the label of the output membrane, the one where the result is obtained.

A transition between two configurations C_1, C_2 of Π is denoted by $C_1 \Longrightarrow C_2$, and the set of numbers generated by Π is denoted by $N(\Pi)$.

The rules of the arbitrary form $u \to v$ are said to be *cooperative*, if $u \in O$, then the rule is called *non-cooperative* (it corresponds to context-free rules in a grammar); an intermediate case is that of *catalytic* rules, which are of the form $ca \to cv$, where $c \in O$ is a catalyst, assisting the object $a \in O$ to get transformed into $v \in O^*$. When applying a rule $u \to v$, the objects from u are consumed and those from v are produced.

An *antiport* rule is of the form $(u, out; v, in)$ with $u, v \in O^*$; using such a rule (associated with a membrane i) means to move the multiset u outside membrane i, simultaneously with bringing the multiset v inside the membrane. If one of the multisets u, v is empty, then the rule becomes a *symport* one.

We do not give here further technical definitions or notations; the interested reader can consult any of the titles indicated in the end of the Introduction, especially the Handbook [8].

However, we mention informally a series of notions and of further classes of P systems. There are many possibilities to extend the previously introduced computing device and its functioning. Instead of counting objects in a compartment, we can consider as the result of a computation the sequence of objects sent to the environment (this is the so-called, *external output*), hence a P system can then generate a language. A language is obtained also if we follow the *trace* of a special objects across membranes. Then, we can use a (symport/antiport) P system in the accepting mode: the objects entering the system from the environment are arranged in a string and we say that the string is accepted if the computation halts. In the case of tissue P systems, the objects can evolve inside membranes by multiset rewriting rules and can pass from a membrane to another one by antiport rules. The communication channels among membranes are hence implicitly defined by the provided rules for communication; a more complex case is that of *population* P systems, where there also are rules for establishing channels between cells and for destroying them. Besides rules for handling objects, we can also have rules for changing the membrane structure. We mentioned division, creation, and dissolution rules, exo- and endocytosis, but there also are separation, budding, gemmation rules. Observe the biological inspiration, although abstracted in a way which brings us far from biology – in their initial forms, P systems were not meant to be used as models with a biological relevance. The objects can be described by symbols, as above, but they can also have a structure, for instance, described by strings (processed by string operations, such as rewriting, DNA splicing, replication, insertion-deletion), or even more complex, such as 2D arrays, trees, etc. A special case is that of *numerical* P systems, where numerical variables are placed in the regions of a cell-like membrane structure, evolving by means of *programs*, composed of a *production function* (e.g., a polynomial), and a *repartition protocol*; in each compartment, the local variables are subject of a local production function, and the value of this function is distributed among the variables in that region and in the neighboring regions according to the repartition protocol (e.g., proportionally with given numbers, part of the program). The model, somewhat inspired from economics, can both generate sets of numbers, but also compute functions of several variables, a situation which is completely different from the generative-accepting functioning of usual object-based P systems. An interesting variant is that of P systems with objects bound on membranes (as actually is the case with many chemicals in a cell), and then with the rules evolving at the same time objects which are free inside regions and these fixed objects.

Finally, let us mention the so-called *spiking neural P systems* (SN P systems), where membranes (representing neurons) are placed in the nodes of a graph, whose links represent *synapses*, holding several copies of a single object, corresponding to a *spike*; the spikes evolve by rules which first check the contents of the neuron (by means of a regular expression), consume a number of spikes and produce a number of spikes, which are sent, immediately or with a delay, to all neurons to which a synapse goes from the neuron where the rule was used. The spikes sent to the environment by a designated output neuron form the *spike train* produced by the system; numbers or strings can be associated with a spike train, hence again a generative device is obtained.

Up to now, we mentioned only the *generative* mode (corresponding to grammars) of using a P system. A dual case (corresponding to automata) is the *accepting* mode: a number is introduced in a system, e.g., as the multiplicity of a specified object in a specified compartment, and the number is accepted if the computation halts. Strings can also be recognized, by bringing their symbols, one by one, in a system (e.g., in a symport/antiport one), with the string accepted if the computation halts.

In all cases, we can also use a P system as a *decidability* machine: a decision problem (with YES/NO answer) is introduced in the system, encoded in a specified way in the form of a multiset, and the system says whether the problem (actually, its instance introduced in the initial configuration) has an affirmative answer by halting or by sending a special object `yes` into the environment. This is the usual way of investigating the computational complexity of P systems (the time or the space needed to solve a class of decidability problems).

Most classes of P systems are computationally complete, equivalent with Turing machines (one also says that they are universal), even in restricted cases: small number of membranes, using only catalytic rules (with at least two catalysts: the power of one catalyst P systems is still open), symport/antiport rules of reduced sizes, SN P systems of restricted forms, etc. Similarly, many classes of P systems able to create an exponential working space in a linear time (the typical case is that of P systems using membrane division, also called *with active membranes*) can solve **NP**-complete problems (sometimes even **PSPACE** problems) in a polynomial time. The literature of MC abounds in results of these types.

An important part of the research in MC deals with applications. Using P systems for modeling processes taking place in a cell or in complexes of cells, such as populations of bacteria, is expected; the model starts from biology, hence it is natural to return to biology. Several features make P systems attractive for the biologist (especially in comparison with the models based on differential equations): the direct connection with the biochemistry, which also means a high understandability, the multicompartmental structure, the easy scalability, the intrinsic discrete nature of the model, the easy programmability, the possibility to attach probabilities (reaction rates, stoichiometric coefficients) to the evolution rules, the emergent behavior of a P system (the overall evolution is not at all a "sum" of the parts evolution). All these applications are based on simulation programs (there are several such programs available – see the webpage of the domain, mentioned in the bibliography of the Introduction, [9]). Most of them run on the usual sequential computers, but there also are attempt to implement P systems on dedicated hardware, clusters and grids, on parallel hardware (such as NVIDIA graphical cards). A specialized programming language, *P-lingua*, was also elaborated.

Also somewhat expected are the applications in modeling and simulating ecosystems (we have "membranes" where several agents interact, like the chemicals in a cell). Not so expected however are the applications in approximate optimization (distributed evolutionary computing), computer graphics (following the style of L systems based graphics, but also recent attempts to process images in the parallel framework of P systems), while the recent applications of numerical P systems in controlling mobile robots is completely unexpected.

Problems and research topics about most of these issues will be found in the sections below. Of course, the previous bird's eye view about MC is not enough to technically address these problems, many details were omitted or given in an approximate way, but at least the reader can have an image of this research area, of its many branches, of the richness of results and applications, but also of the fact that many issues still wait for clarifications. The frontiers of MC are still moving, after more than 13 years since this research area was initiated, it still can be considered as an "Emergent Research Front in Computer Science", as it was called already in 2003 by Thomson-Reuters Institute for Scientific Information, ISI, with [1] considered a "fast breaking paper", see `http://esi-topics.com`.

References

1. Gh. Păun: Computing with membranes. *J. Computer and System Sciences*, 61, 1 (2000), 108–143.

2 Some General Issues

Jacob Beal

BBN Technologies, Cambridge, MA, USA
`jakebeal@bbn.com`

Comment. Jacob Beal was one of the invited speakers at CMC 12 (title of talk: "Bringing Biology and Engineering Together with Spatial Computing"). After the meeting, he was asked to express his thoughts about MC, taking into account that he comes from outside the MC community, more importantly, from applied computer science. What follows is part of an e-mail message he sent to M.Gh. around the end of August 2011.

Required Notions: general knowledge of MC, membrane structure, multiset processing, distribution, parallelism

With regards to my thoughts on directions for the membrane computing community, I think there is something very interesting and unique about the combination of chemical, compartmentalized, and tree-structured computation that P systems give access to. But I think that it is important to try to articulate what that is and why it is important.

In particular, the questions that I might pose would be:

- What are the most important research questions for membrane computing?

- What does membrane computing have to offer researchers who are not in the field of membrane computing?

  More specifically:

- What should other computational theorists learn from the family of P systems computational models?
- What is the practical advantage of P systems models over their competitors in biological modeling or other fields?
- How might P systems models be applied to improve representations or architectures for parallel computing?
- What is quantitatively advantageous about SN P systems over other spiking models?
- How can P systems inform the theory or design of distributed algorithms?

I do not expect that any of these questions will have any one answer – in fact, I am sure that many researchers in the field will have wildly different answers. But every researcher should have clear and concise answers that they can make a good case for.

For my own part, I think that the most important research questions are:

1. How can distributed systems notions like self-stabilization be applied to P systems?
2. What consequences does the P systems model have for conventional computing?
3. What sort of complex P systems computations can be generated from high level programming languages, and what sort of languages fit best with P systems for various purposes (e.g., biological modeling, networking)?

Those priorities, however, are of course a consequence of my own research interests and biases, and I expect that others would have different answers: the important thing is the discussion of reasons.

3 The Power of Small Numbers

Artiom Alhazov

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Chişinău, Republic of Moldova, and

Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Milano, Italy
artiom@math.md, aartiom@yahoo.com

Comment. Artiom Alhazov was also an invited speaker at CMC 12; his talk, "Properties of Membrane Systems", is cited in the references below and can be helpful in clarifying some of the notions mentioned in the following problems.

Several problems related to the optimality of certain parameters appearing in characterization of the computing power of various classes of P systems (symport/antiport, insertion-deletion, with active membranes) and of their efficiency; in particular, questions about the languages described (in the external mode) by P systems are formulated.

Required Notions: symport/antiport, external output, active membranes, minimal parallelism, insertion-deletion

Note: in case the underlying definitions are not clear, all bibliography items include URLs of the associated publications (freely accessible .PDF files or springerlink references). This made it possible to formulate the problems more concisely.

3.1 Minimal Parallelism and Number of Membrane Polarizations (2006)

It is known, [1, 2] that under minimal parallelism, P systems with polarized active membranes can solve intractable problems in a polynomial number of steps, even without non-elementary membrane division and without membrane creation. However, the best known results deal with P systems using 6 (six!) polarizations, or 4 polarizations if non-standard rule types (evolution rules are applied sequentially and may change the polarization) are used. Are these numbers optimal?

3.2 Membrane Systems Language Class (2010)

A fundamental family of languages is still not characterized: languages generated (in the sense of external output) by non-cooperative membrane systems. It is known, [5, 4] that the best known lower bound for $LOP(ncoo, tar)$ is $REG \cdot \mathtt{Perm}(REG)$ (strict inclusion), while the best known upper bound is $CS \cap SLIN \cap \mathbf{P}$. An example of a difficult language in this family is

$$\{\ \texttt{Perm}((abc)^{2k_0})\texttt{Perm}((a'b'c')^{2k_1})\cdots\texttt{Perm}((abc)^{2k_{2t}})\texttt{Perm}((a'b'c')^{2k_{2t+1}})$$
$$\mid k_0 = 1,\ 0 \leq k_i \leq 2k_{i-1},\ 1 \leq i \leq 2t+1,\ t \geq 0\}.$$

Open questions concerning comparison of the P systems language family with particular language families and concerning particular closure properties are also formulated in the above mentioned papers.

3.3 Dynamical Properties (2011)

It is well-known, e.g., that catalytic P systems are computationally complete, while deterministic catalytic P systems are not.

In [3], an overview of a number of dynamical properties of P systems is given, the most important one being determinism. In particular, five variants are recalled where nondeterminism seems an essential source of the computational power (although, as far as we know, no formal proof of power separation has been obtained), with informal justification for the word "seems":

1. P systems with active membranes, where except membrane separation, the rules are non-cooperative and the membrane structure is static (solving SAT).
2. Non-cooperative P systems with promoters or inhibitors of weight not restricted to one (universality).
3. Minimal combinations of alphabet size/number of membranes or cells (universality).
4. P systems without polarizations (universality).
5. Conditional uniport (universality).

The open question is, for any of the variants above, to formally prove that determinism decreases the computational power of the corresponding systems (as it is in the case of catalytic systems).

The post-proceedings version of [3] (i.e., the version appearing the LNCS volume) also proposes to study 6 new formal properties inspired by self-stabilization concept.

3.4 Exo-Insertion/Deletion (2011)

This is the only open problem in this list that concerns P systems with string objects. Consider P systems with string objects and operations of right or left insertion or deletion of given strings. The problem is to find a characterization of the power of P systems with exo-insertion of weight one and exo-deletion of weight one without contexts ("exo" means leftmost or rightmost).

There exist the following partial results:

- Not computationally complete if operations (even both with weight two) are performed anywhere in the string.
- Computationally complete if insertion has weight two.
- Computationally complete if deletion has weight two.

- Computationally complete for tissue P systems.
- Computationally complete if deletion has priority over insertion (even without deletion on the right).
- The lower bound is the family of regular languages (even with all operations on one side).

3.5 Symport-3 in One Membrane (2005)

Reaching for universality by moving objects across a single membrane leads to interesting combinatorial questions. While antiport roughly corresponds to rewriting, symport does not provide such an intuitive counterpart, although it remotely resembles insertion/deletion or vector addition.

It is well known that the minimal size of symport rules for the universality in one membrane is 3, [6]. The computational completeness is achieved there with 7 additional objects in the skin. It is not difficult to see that at least one object is necessary, or only finite sets are generated.

Indeed, the only way to increase the number of objects is to send something out, so that something comes back in, bringing something else. Generating any infinite set means that such a procedure must be iterated. Hence, sending all objects out cannot lead to halting.

Therefore, the lower bound for $LOP_1(sym_3)$ is N_7RE, while the upper bound is $N_1RE \cup NFIN$. It is an open problem to bridge (or at least decrease) the gap by investigating what sets containing numbers smaller than 7 can be generated.

References

1. A. Alhazov: Minimal parallelism and number of membrane polarizations. *The Computer Science Journal of Moldova*, 18, 2 (2010), 149–170.
   `http://www.math.md/publications/csjm/issues/v18-n2/10284/`
2. A. Alhazov: Minimal parallelism and number of membrane polarizations. *Preproc. Seventh International Workshop on Membrane Computing* (H.J. Hoogeboom et al., eds.), WMC7, Lorentz Center, Leiden, 2006, 74–87.
   `http://wmc7.liacs.nl/proceedings/WMC7Alhazov.pdf`
3. A. Alhazov: Properties of membrane systems. *Preproc. Twelfth International Conference on Membrane Computing* (M. Gheorghe et al., eds.), CMC12, Fontainebleau, 2011, 3–14.
   `http://cmc12.lacl.fr/cmc12proceedings.pdf`
4. A. Alhazov, C. Ciubotaru, S. Ivanov, Yu. Rogozhin: The family of languages generated by non-cooperative membrane systems. *Membrane Computing. 11th International Conference*, CMC 2010, Jena, 2010. Revised Selected Papers (M. Gheorghe et al., eds.) LNCS 6501, Springer, 2011, 65–80.
   `http://www.springerlink.com/content/gt07j477k2020417/`
5. A. Alhazov, C. Ciubotaru, S. Ivanov, Yu. Rogozhin: Membrane systems languages are polynomial-time parsable. *The Computer Science Journal of Moldova*, 18, 2 (2010), 139–148.
   `http://www.math.md/publications/csjm/issues/v18-n2/10282/`

6. A. Alhazov, R. Freund, Yu. Rogozhin: Computational power of symport/antiport: History, advances and open problems. *Membrane Computing, 6th International Workshop* (R. Freund et al., eds.), WMC 2005, Vienna, Revised Selected and Invited Papers, LNCS 3850, Springer, 2006, 1–30.
`http://springerlink.metapress.com/index/f500782x34331741`
7. A. Alhazov, A. Krassovitskiy, Yu. Rogozhin: Circular Post machines and P systems with exo- insertion and deletion. *Preproc. Twelfth International Conference on Membrane Computing* (M. Gheorghe et al., eds.), CMC12, Fontainebleau, 2011, 63–76.
`http://cmc12.lacl.fr/cmc12proceedings.pdf`

4 Polymorphic P Systems

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

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova, Chişinău, Republic of Moldova
`{artiom,rogozhin,sivanov}@math.md`

[2] University of Academy of Sciences of Moldova
Faculty of Real Sciences, Chişinău, Republic of Moldova

[3] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione, Milano, Italy
`aartiom@yahoo.com`

Polymorphic P systems are briefly introduced (systems where the evolution rules are produced dynamically, during the computation) and several problems about their power and efficiency are formulated.

Required Notions: cell P systems, active membrane, complexity

Polymorphic P systems introduce a new feature into membrane computing. This time the inspiration does not come from biology, but rather from conventional computing and namely from von Neumann architecture. The point is in not fixing the rules in the structural description of the P system, but rather storing them as contents of membranes. This new construction has not yet been studied properly; very little is known about the computational power of polymorphic P systems.

Formally, we define a *polymorphic P system* as a tuple

$$P = (O, T,\ \mu,\ w_s, w_{1L}, w_{1R}, \dots, w_{mL}, w_{mR},\ \varphi,\ i_{out}).$$

The set O is a finite alphabet, $T \subseteq O$ is the set of output objects, μ is a tree structure consisting of $2m + 1$ membranes bijectively labeled with the elements of $H = \{s\} \cup \{iL, iR \mid 1 \leq i \leq m\}$. The skin membrane is labeled with s. It is required that the parent membrane of iL is the same as the parent membrane of

iR for all $1 \leq i \leq m$. The string w_h, $h \in H$, is the initial content of the membrane with label h. The label i_{out} indicates the region where the output of the system will be read from. We will describe the mapping φ later on.

Observe that the description of the system does not include any rule. Instead, the contents of the membranes with labels iL and iR are interpreted as the left-hand side and right-hand side of the rule i respectively. At every step, the rules are applied in the usual way. As a result of application of the rule i, the right-hand side of the rule (the content of iR) is injected into $\varphi(i)$. The latter mapping is defined as follows: $\varphi : \{1, \dots, m\} \to Tar$, $Tar = \{in_j \mid j \in H \text{ is an inner membrane of } p\} \cup \{out, here\}$, where $p \in H$ is the label of the membrane containing the rule i (the membranes iL and iR). For further information we refer the reader to [1].

Polymorphic P systems have not yet been explored sufficiently well. In the following paragraphs we list some open problems which we find of interest.

- *Solve hard problems.* It has been shown that polymorphic P systems can solve certain problems faster than any other P system model (for example, they generate n^2 in $O(1)$ and generate 2^{2^n} in $O(n)$). So far, only relatively simple problems were considered, but we believe that the polymorphic model has the potential to facilitate solving much harder problems. For example, possibilities to find the Gröbner basis using polymorphic P systems are currently being considered.

- *Characterize problems which may be solved faster.* A more general question, on the other hand, is to define the class of problems which can be solved more efficiently using polymorphic P systems. It has been observed that, for multiplication, linear speed-up was introduced; a much more systematic research in this direction is necessary. In particular, it is unclear whether it is possible to use the polymorphism to construct exponential workspace for solving intractable problems in polynomial time.

- *Polymorphic P systems with active membranes.* Polymorphic P systems are a fairly simple model at the moment. This means, in particular, that certain extensions are possible. We would like to particularly stress the perspectives of considering polymorphic P systems with active membranes, where the membrane structure itself does not stay constant. Such a combination is a very powerful one, therefore it is important to establish some restrictions which will define an as simple as possible, yet sufficiently powerful, construct.

- *The power of the most restricted variant.* Another way to explore polymorphic P systems is characterizing the power of models with the minimal number of additional ingredients (non-cooperative rules, no rules with empty left-hand side, no target indications). In [1] it is shown that even this model can easily achieve superexponential growth; it is important to know how powerful polymorphism on its own is.

- *Self-assembly.* Finally, we make the observation that rules in polymorphic P systems may be treated as results of interaction of couples of initially independent membranes, which have gained additional capabilities by connecting to each other. The whole polymorphic P system may be treated as a stage in the process of interaction of membranes in a system of membranes. This brings about, in particular, the question of self-assembly of membrane structures.

References

1. A. Alhazov, S. Ivanov, Yu. Rogozhin: Polymorphic P systems. *11th International Conference on Membrane Computing* (M. Gheorghe et al., eds.), CMC 2010, Jena, Germany, LNCS 6501, Springer, Berlin, 2010, 81–94.

5 P Colonies and dP Automata

Erzsébet Csuhaj-Varjú

Department of Algorithms and Their Applications
Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
csuhaj@inf.elte.hu

Required Notions: tissue-like P system, P colony, dP automaton

5.1 P Colonies

P colonies are variants of very simple tissue-like P systems, modeling a community of very simple cells living together in a shared environment (for basic information see [8]).

In the basic model, the cells (or 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 functioning 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 is present in the environment. 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 of such 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. At the beginning of the computation, performed by a given P colony of capacity k, the environment contains arbitrarily many copies of a distinguished symbol e, called the environmental symbol (and no other symbols); furthermore, each cell contains k copies of e. When a halting configuration is reached, that is, when no more rules can be applied, the result of the computation is read as the number of certain types of objects present in the environment.

P colonies have been extensively examined during the years. It was shown that these simple constructs are computationally complete computing devices even with very restricted size parameters and with other syntactical or functioning restrictions. Several extensions of the model have already been investigated as well: P colonies with dynamically varying environment (eco-P colonies) [1] or PCol automata [2], constructs where the behavior of the cells is influenced by direct impulses coming from the environment step-by-step. In the case of a PCol automaton a tape with an input string is given with the P colony, i.e., the model is augmented with a string put on an input tape to be processed by the P colony.

Except PCol automata, P colonies have been considered as generating devices, but the construct can also be considered as a (multiset) *accepting device* (called *accepting P colony* or *P colony acceptor*), possibly working in an automaton-like fashion as well. In the following we propose problems and problem areas in this direction.

To define such a model, suppose that we have a P colony Π of capacity k and initialize the environment with a given finite multiset of symbols M where each symbol is different from the environmental symbol e. Let also consider an initial configuration, i.e., let us dedicate an initial state to any cell and let us distinguish a set of accepting configurations. Then, we say that M is accepted by Π, if after performing a finite computation (in some computation mode) the environment consists of only symbols e.

It is easy to see that we may consider several variants of this model. For example,

- we can limit the number of symbols in the environment (not necessarily with a finite constant, but with some function of the size of the P colony) and study the computational power of these systems with limited workspace for the computation,
- we can consider the multisets in the environment during the computation as permutations of words (or map them to words in some other way) being on the input tape of an automata and study the relation of these constructs and classical automata;
- we can map the sequences of multisets of objects entering each cell during the computation to words being on the input tape of a multitape or multihead automata and describe the correspondence between these constructs and the classical multitape or multihead automata variants.

By introducing double alphabets as in the case of dP automata for describing two-way multihead finite automata ([3]), automata with two-way motion of heads can also be interpreted in the framework of accepting P colonies.

The concept of accepting P colonies can be extended in some other manners as well. For example, we do not fix the number of cells in the P colony in advance but it is determined by the number of non-environmental symbols in the environment at the beginning. Spatial P colonies can also be defined. In this case spatial parameters are added to the cells and a neighborhood relation among the components is given; a cell can import only such symbols from the environment which were issued by its neighbors (are placed in its own environment).

Accepting P colonies can be related to cellular automata as well. One natural idea is to define P colonies corresponding to one-way cellular automata, which are linear arrays of identical copies of deterministic finite automata, called cells, working synchronously at discrete time steps. Each cell is connected to its immediate neighbors to the right. The cells are identified by positive integers. The state transition depends on the current state of a cell itself and the current state of its neighbor. An input word is accepted by a one-way cellular automaton if at some step in the course of the computation the leftmost cell enters an accepting state.

A particular variant of one-way cellular automata is the one where only a fixed number, say k, cells are given. This works similarly to the unrestricted case, but the input is processed in a different manner, namely, the input is not given at the beginning, but it is processed by the rightmost cell, symbol by symbol. Since the neighborhood can be defined in P colonies with emitting special symbols (signals) in the environment and any cell in the P colony may have only a finite number of configurations (states), the reader may observe that the two computational models, the accepting P colony and the k-cell one-way cellular automaton are strongly related.

Obviously, more general cellular automata models can also be described by P colony acceptors. For example, the above extension of the concept of P colonies where the number of cells is determined by the number of initial non-environmental symbols can correspond to the unrestricted case. We can also model d-dimensional cellular automata ($d \geq 1$) by defining the neighborhood relation between cells of P colonies in an appropriate manner. Cellular automata theory has been a highly elaborated field of nature-motivated, parallel computing (see, for example, [5], [6], [7]), thus by building bridges between P colony theory and cellular automata theory, many interesting problems can also be studied.

5.2 dP Automata

In addition to comparing accepting P colonies to variants of classical automata, we may explore the differences and similarities between these constructs and (finite) dP automata as well. A detailed study in this direction would also help in better understanding the nature of these two constructs.

P automata are variants of antiport P systems accepting strings in an automaton-like fashion (for a summary on P automata, see Chapter 6 of [8]). The notion of a distributed P automaton (dP automaton in short) was introduced in [9]. Such a system consists of a finite number of component P automata which have their separate inputs and which also may communicate with each other by means of special antiport-like rules. A string accepted by a dP automaton is obtained in [9] as the concatenation of the strings accepted by the individual components during a computation performed by the system. A dP automaton is called finite if it has only a finite number of different configurations.

The computational power of dP automata was studied in [9], [4], [10], and [11]. In [3] a connection between finite dP automata and non-deterministic multi-head finite automata was explored. It was shown that the language of a non-deterministic one-way multi-head finite automaton and the language of a non-deterministic two-way multi-head finite automaton can be obtained as so-called weak agreement language or strong agreement language of a one-way, i.e., a usual finite dP automaton, and a two-way finite dP automaton.

The reader may easily observe that finite dP automata, P colony acceptors and cellular automata are closely related concepts. Their comparative study would be a promising and very useful area in P systems theory.

Acknowledgement. Work supported in part by the Hungarian Research Fund "OTKA", project K75952.

References

1. L. Cienciala, L. Ciencialová: Eco-P colonies. *Proc. WMC 2009*, LNCS 5957 (Gh. Păun et al., eds.), Springer, 201–209.
2. L. Ciencala, L. Ciencialová, E. Csuhaj-Varjú, Gy. Vaszil: PCol automata: Recognizing strings with P colonies. *Proc. BWMC 2010, Sevilla, 2010* (M.A. Martínez-del-Amor et al., eds.), Fénix Editora, Sevilla, 2010, 65–76.
3. E. Csuhaj-Varjú, Gy. Vaszil: A connection between finite dP automata and multi-head finite automata. *Proc. Twelfth International Conference on Membrane Computing*, Fontainebleau, 23-26 August, 2011 (M. Gheorghe et al., eds.), 109–126.
4. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest Univ. Math.-Informatics Series*, 63, 2009, 5–22.
5. M. Kutrib: Nature-based problems in cellular automata, *Proc. CiE 2011*, LNCS 6735, Springer, 2011, 171–180.
6. M. Kutrib, J. Lefevre, A. Malcher: The size of one-way cellular automata. *Proc. Automata 2010: Discrete Mathematics and Theoretical Computer Science*, DMTCS, 2010, 71–90.
7. M. Holzer, M. Kutrib: Cellular automata and the quest for nontrivial artificial self-reproduction. *Proc. Conf. on Membrane Computing, CMC 2010*, LNCS 6501, Springer, 2010, 19–36.
8. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*, Oxford Univ. Press, 2010.
9. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems, *International Journal of Computers, Communication & Control*, V(2), 2010, 238–250.
10. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Rainbow of Computer Science* (C.S. Calude et al., eds.), LNCS 6570, Springer, 2011, 102–115.

11. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Sci.*, 431 (2012), 4–12.

6 Spiking Neural P Systems

Linqiang Pan, Tao Song

Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology, Wuhan, Hubei, China
lqpan@mail.hust.edu.cn, songtao608@gmail.com

Applications of spiking neural P systems are proposed and some problems related to such applications are formulated.

Required Notions: spiking neuron, SN P system, spiking neural network

Spiking neural P systems (SN P systems, for short) were introduced in [4] as a class of distributed and parallel computing models inspired by spiking neurons. In an SN P system, the *neurons* are placed in the nodes of a directed graph. The content of each neuron consists of a number of copies of a single object type, called the *spike*. Each neuron contains a number of *firing* and *forgetting rules*. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes) which are accumulated at the target cells. The applicability of each rule is determined by checking the content of the neuron against a regular set associated with the rule. A forgetting rule removes a specified number of spikes from the neuron. In each time unit, if a neuron can use some of its rules, firing or forgetting, then one of the rules must be used. The rule to be applied is nondeterministically chosen.

One of the neurons is designated as the output neuron of the system, and its spikes are also sent to the environment; their sequence is called *the spike train* generated by the system. Several *results of a computation* can be defined associated with the spike train (strings or numbers).

SN P systems use individual spikes allowing to incorporate spatial and temporal information in computation, which corresponds to the fact that neurons use spatial and temporal information of incoming spikes to encode their message to other neurons, where the number and timing of spikes matters. In the above sense, SN P systems fall into the third generation of neural network models [6].

Many computational properties of SN P systems have been studied (but many of them raise further research topics, but we do not refer to them here). SN P systems were proved to be computationally complete as number computing devices [4], language generators [1, 2], and function computing devices [8]. SN P systems were also used to (theoretically) solve computationally hard problems in a feasible time [5, 7]. In contrast with the relatively rich theoretical results, the practical applications of SN P systems are few (although some attempts are already made, e.g., Hebbian learning in the framework of SN P systems [3]). However, as a representative of the third generation of neural network models, spiking neural networks (SNNs) could have very hands-on applications such as speech recognition, learning, associative memory, function approximation (see, e.g., *Information Processing Letters*, 95, 2005), and have proved to be useful in neuroscience. It is interesting to move the SN P systems investigations towards applications. In the following, we list some problems which we find of interest.

- In SN P systems, the use of spike timing information is based on regular expressions, which can be considered as an integrate-and-fire scheme. The scheme of regular expressions is quite different from the traditional ones, such as the sigmoidal scheme. What is the advantage of the scheme of regular expressions from the application point of view? Can the two schemes (the regular expression and the sigmoidal one) be related?
- What ingredients can be added to SN P systems for practical applications (maybe, noise, randomness)?
- What are the specific real world problems where SN P systems have a practical advantage over other SNNs?
- How can some variants of SN P systems be designed such that they would deal with features of more biological plausibleness?

References

1. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7 (2008), 147–166.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75 (2007), 141–162.
3. M.A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez: Hebbian learning from spiking neural P systems view. *Proc. WMC9 2008* (D. Corne et al., eds.), LNCS 5391, Springer, Berlin, 2009, 217–230.
4. M. Ionescu, G. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2006), 279–308.
5. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science*, 411 (2010), 2345–2358.
6. W. Maass: *The Third Generation of Neural Network Models.* Technische Universitat Gräz, 1997
7. L. Pan, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with neuron division and budding. *Science China Information Sciences*, 54 (2011), 1596–1607.
8. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90 (2007), 48–60.

7 Control Words Associated with P Systems

Kamala Krithivasan[1], **Gheorghe Păun**[2], **Ajeesh Ramanujan**[1]

[1]Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai, India
kamala@iitm.ac.in

[2]Institute of Mathematics of the Romanian Academy
Bucharest, Romania, and

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
gpaun@us.es

Ways to associate a control word with a computation in a P system are proposed and some of the problems which are natural to be investigated in this respect are mentioned.

Required Notions: Szilard language, Chomsky hierarchy, cell P system, SN P system, parallelism

Control words are almost never considered in membrane computing – actually, we know no paper dealing with this issue, although generating or recognizing languages are central research topics (with the languages identified by the sequence of symbols entering or leaving a P system, or by traces of certain symbols in their passage across membranes). The reason is the fact that in the same step of a computation several rules are used, possibly with several labels, hence the control word is not clearly defined. On the other hand, a sort of bidimensional control word was introduced already during the first BWMC, in [1], under the name of *Sevilla carpet*, as a way to describe the rules used in a computation and their multiplicity in each step, but not as a way to define a control language associated with the computations in a P system.

A possible solution to the above difficulty is to consider a sequence of multisets of labels, those labels associated with all rules applied in a given step. Then, a string of symbols can be obtained following the ideas also used for accepting P systems: take a function from multisets to strings and build the string(s) obtained by concatenating the strings associated with the multisets. For instance, all permutations of the labels in a multiset can be considered, as in [3], or only one specific string (maybe a symbol) associated with the multiset, like in [2].

Another idea was recently introduced in [4], starting from the following restriction: all rules used in a computation step should have the same label, or they can also be labeled with λ.

The definition in [4] is given for SN P systems, but it works for any type of P systems, not only for SN P systems.

Indeed, let us consider a P system Π, of any type, with the total set of rules (the union of all sets of rules associated with compartments, membranes, neurons – as it is the case) denoted with R. Consider a labeling mapping $l : R \to B \cup \{\lambda\}$, where B is an alphabet. We consider only transitions $s \Longrightarrow_b s'$, between configurations s, s' of Π, which use only rules with the same label b and rules labeled with λ. We say that such a transition is *label restricted*. With a label restricted transition we associate the symbol b if at least one rule with label b is used; if all used rules have the label λ, then we associate λ to this transition. Thus, with any computation in Π starting from the initial configuration and proceeding through label restricted transitions we associate a (control) word. Consider also a criterion $\mathtt{C}$ of the correct termination of a computation (e.g., halting or reaching a configuration from a given set F of final configurations, or both of these, etc.) The language of control words associated with all label restricted computations in Π which are correctly terminated (with respect to $\mathtt{C}$) is denoted by $Sz_{\mathtt{C}}(\Pi)$ (with *Sz* coming from *Szilard*, as usual in language theory).

Now, a series of natural problems can be formulated: investigate the languages of control words for (i) various classes of P systems, with (ii) various criteria $\mathtt{C}$, in particular, (iii) allow only transitions which use at least a rule labeled by $b \in B$. When λ transitions are accepted, characterizations of RE languages are expected, but when each step produces a symbol, there is no possibility for "hidden work", the computation has the same length as the control string, so that the generated language is recursive. In this latter case the comparison with language families in Chomsky hierarchy is of interest (with the conjecture that languages of the forms $\{xx \mid x \in V^*\}$, $\{xx^R \mid x \in V^*\}$, where $card(V) \geq 2$ and x^r is the mirror image of x, cannot be obtained as the language of control words of a P system.

In particular, the languages $Sz_{\mathtt{C}}(\Pi)$ can be associated with SN P systems, with or without anti-spikes. We expect interesting (language theory) results in this research area.

References

1. 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.
2. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems. *Membrane Computing, International Workshop, WMC-CdeA, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers* (Gh. Păun et al., eds.), LNCS 2597, Springer, 2003, 219–233.
3. M. Oswald: *P Automata*, PhD Thesis, TU Viena, 2003.
4. A. Ramanujan, K. Krithivasan: Control words of spiking neural P systems. Paper in preparation.

8 Speeding Up P Automata

György Vaszil

Department of Computer Science, Faculty of Informatics
University of Debrecen, Hungary
vaszil.gyorgy@inf.unideb.hu

The issue of efficient parallelization of languages with respect to dP automata is discussed (especially, the dependence on the multiset-to-strings functions which are used to define the input language).

Required Notions: Regular language, context-sensitive language, P automata and dP automata (accepted multiset sequence, input mapping, accepted language)

This section deals with the possibility of speeding up P automata computations (in a similar sense as a linear speedup for Turing machines is possible), a problem which is important from the point of view of the efficiency of the parallelization of P automata computations with distributed P automata.

A *P automaton*, introduced in [2], is an antiport P system placed in an environment, from where a sequence of input multisets is read during the computation. A multiset sequence is accepted, if the computation ends in an accepting configuration, and the accepted multiset sequence is interpreted as a string (a sequence of symbols) using a so called input mapping $f : V^* \to 2^{T^*}$ where T is a finite alphabet and V is the object alphabet of the P automaton. (We assume that f is nonerasing, that is, $f(u)$ is the empty word for some multiset $u \in V^*$, if and only if u is empty.) The language accepted by a P automaton Π with respect to f is defined as $L(\Pi, f) = \{f(v_1) \dots f(v_s) \mid v_1, \dots, v_s$ is an accepted multiset sequence of $\Pi\}$.

It is obvious that the choice of the mapping f has a great influence on the accepting power of the P automaton, so let us take a closer look at the mappings we can use.

Let $f : V^* \to 2^{T^*}$, and (1) let us denote f with f_{perm}, if and only if $V = T$, and for all $v \in V^*$, we have $f(v) = \{u \mid u$ is a permutation of $v\}$. Moreover, (2) we say that $f \in$ TRANS, if and only if for any $v \in V^*$, we have $f(v) = \{w\}$ for some $w \in T^*$ which is obtained by applying a finite transducer to the string representation of the multiset v (as w is unique, the transducer must be constructed in such a way that all string representations of the multiset v as input result in the same $w \in T^*$ as output, and moreover, as f should be nonerasing, the transducer produces a result with $w \neq \lambda$ for any nonempty input).

Let us recall from [6] that there are simple linear languages which cannot be accepted by P automata with f_{perm}, for example $L = \{(ab)^n(ac)^n \mid n \geq 1\}$ is such a language. On the other hand, the class of languages accepted with f_{perm} also contains non-context-free context-sensitive languages ($\{a^nb^nc^n \mid n \geq 1\}$ for example), which means that it is incomparable with the class of linear and of context-free languages. (Although it contains all regular languages, see [3].) In contrast to these results, systems with input mappings from the class TRANS characterize the class of context-sensitive languages (see [1] for details).

The notion of distributed P automaton (dP automaton) was introduced in [5] to incorporate a "different kind of parallelism" into P systems: the components of a dP automaton are P automata which process different parts of the input in parallel. The language $L \subseteq T^*$ accepted by a dP automaton consists of words of the form $w_1 w_2 \dots w_k$ where $w_i \in T^*$ are strings accepted by the component Π_i, $1 \leq i \leq k$, during a successful computation. Let $f = (f_1, \dots, f_k)$ be a mapping $f : (V^*)^k \to (2^{T^*})^k$ with $f_i : V^* \to 2^{T^*}$, $1 \leq i \leq k$, being non-erasing, and let $L(d\Pi, f) = \{w_1 \dots w_k \in T^* \mid w_i \in f_i(v_{i,1}) \dots f_i(v_{i,s_i}),\ 1 \leq i \leq k,$ where $v_{i,1}, \dots, v_{i,s_i}$ is an accepted multiset sequence of the component $\Pi_i\}$.

A language is efficiently parallelizable, as defined in [5], if it can be accepted by a dP automaton in "less" computational steps than by any non-distributed P automaton, that is, L is (k, l, m)*-efficiently parallelizable with respect to a class of mappings* F, for some $k, m > 1$, $l \geq 1$, if L can be accepted with a dP automaton $d\Pi$ with k components, such that $L = L(d\Pi, f)$ for some $f \in F$ with $\mathrm{Com}(d\Pi) \leq l$, and moreover, for all P automata Π and $f' \in F$ such that $L = L(\Pi, f')$,

$$\lim_{x \in L, |x| \to \infty} \frac{\mathrm{time}_{\Pi}(x)}{\mathrm{time}_{d\Pi}(x)} \geq m$$

where $\mathrm{time}_X(x)$ denotes the number of computational steps that a device X needs to accept the string x, and where $\mathrm{Com}(d\Pi)$ denotes the maximal amount of communication (measured in some reasonable way, see [5]) between the components of the dP automaton $d\Pi$ during an accepting computation.

By looking at the quotient in the definition above, we might see that a language cannot satisfy the requirement of efficient parallelizability if the dividend (that is, the time that a non-distributed P automaton needs to accept the language) can be made arbitrarily small. This leads us to the problem of the possibility of speeding up P automata computations.

Recall that for Turing machines, a linear speedup is always possible by appropriately encoding the contents of the worktapes, but as the input usually has to remain in its original form on the input tape, the resulting time complexity cannot be less than the length of the input word (see, for example, [4]). Such a lower bound does not necessarily exist in the case of P automata, while the input itself is also "encoded" by the input mapping, so using different mappings, it might be possible to "read" the same word in several different ways, possibly also in different numbers of computational steps.

To demonstrate this, let us recall from [8] that for any regular language L and constant $c > 0$, there exists a P automaton Π such that $L = L(\Pi, f)$ for some $f \in$ TRANS, and for any $w \in L$ with $|w| = n$ it holds that $\mathrm{time}_{\Pi}(w) \leq c \cdot n$. This, as we outlined above, implies that there are no efficiently parallelizable regular languages with respect to the class of input mappings TRANS. The situation is different, however, if instead of an input mapping from the class TRANS, we consider f_{perm}. There are regular languages (called "frozen" in [7]) where the order of no two adjacent symbols can be exchanged, thus, each of them has to be read in different computational steps, which means that the computation of the P automaton cannot be shorter than the length of the input. Thus, it is not surprising that there are efficiently parallelizable regular languages with respect to f_{perm}, as shown in [5].

So far all cases of efficient parallelizability were demonstrated with respect to the input mapping f_{perm}, which makes it interesting to ask the following.

Problem. *Are there languages (over some finite alphabet T) accepted by P automata with object alphabet V which are (k, l, m)-efficiently parallelizable for some $k, m > 1$, $l \geq 1$, with respect to some input mapping $f : V^* \to 2^T$, such that $f \neq f_{perm}$?*

In this context, it would also be interesting to prove the impossibility of efficient parallelization of not just the regular, but also of some more general language classes with respect to a class of input mappings different from f_{perm} (with respect to TRANS for example). To this aim, it would be sufficient to find a general method which (similarly to the case of Turing machines) would enable us to show that with a certain type of input mappings (TRANS for example, as it is the case for regular languages) a linear speedup of P automata is always possible.

Acknowledgement. Work supported in part by the Hungarian Research Fund "OTKA", project K75952.

References

1. E. Csuhaj-Varjú, M. Oswald, Gy. Vaszil: P automata. *The Oxford Handbook of Membrane Computing* (Gh. Păun et al., eds.), Oxford Univ. Press, 2010, chapter 6, 144–167.
2. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems. *Membrane Computing, International Workshop, WMC-CdeA, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers* (Gh. Păun et al., eds.), LNCS 2597, Springer, 2003, 219–233.
3. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest University, Mathematical-Informatics Series*, LVIII (2009), 5–22.
4. J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.
5. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *International Journal of Computing, Communication and Control*, V, 2 (2010), 238–250.
6. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Rainbow of Computer Science* (C. Calude et al., eds.), LNCS 6570, Springer, 2011, 102–115.
7. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Science*, to appear.
8. Gy. Vaszil: On the parallelizability of languages accepted by P automata. *Computation, Cooperation, and Life. Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday* (J. Kelemen, A. Kelemenová, eds.), LNCS 6610, Springer, 2011, 170–178.

9 Space Complexity and the Power of Elementary Membrane Division

Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca, Italy
{leporati, mauri, porreca, zandron}@disco.unimib.it

The title is self-explanatory – it should be mentioned that the Milano team has important contributions related to the (time and space) complexity issue, and mainly open problems related to these results are formulated below.

Required Notions: active membranes, computational complexity theory (including counting problems and oracle Turing machines), space complexity for P systems

Problem 1 (P systems with elementary active membranes). P systems with active membranes [8] are known to be able to solve computationally hard problems in polynomial time by creating exponentially many membranes via division. The most recent result in this area [6] shows that polynomial-time Turing machines having access to an oracle for a **PP** [1] problem (whose computing power includes the polynomial hierarchy [10]) can be simulated by uniform families [4] of P systems with active membranes where the only membranes subject to division are elementary (i.e., not containing further membranes), and no dissolution rules are needed. This result is stated, in symbols, as $\mathbf{P}^{\mathbf{PP}} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$. On the other hand, this kind of P systems cannot solve in polynomial time any problem outside **PSPACE** [9], in symbols $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} \subseteq \mathbf{PSPACE}$. Neither inclusion is known to be proper.

Is $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} = \mathbf{PSPACE}$ or, more generally, is there a precise characterization of $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ in terms of complexity classes for Turing machines?

Problem 2 (Space complexity of P systems with active membranes). A measure of space complexity for P systems has been recently introduced [5] in order to supplement the already rich literature about computational complexity issues in membrane computing [3]. We say that the space required by a P system is the maximal size it can reach during any computation, measured as the sum of the number of membranes and the number of objects. A uniform family $\boldsymbol{\Pi}$ of recognizer P systems [4] is said to solve a problem in space $f\colon \mathbb{N} \to \mathbb{N}$ if no P system in $\boldsymbol{\Pi}$ associated to an input string of length n requires more than $f(n)$ space. Under this notion of space complexity, the class of problems solvable in polynomial space by P systems with active membranes, denoted by $\mathbf{PMCSPACE}_{\mathcal{AM}}$, coincides with **PSPACE** [7]. Furthermore, during the 10th Brainstorming Week on Membrane Computing it was also proved that the problems solvable in exponential space by that variant of P systems and by Turing machines coincide, in symbols $\mathbf{EXPMCSPACE}_{\mathcal{AM}} = \mathbf{EXPSPACE}$.

The techniques used to prove these results do not seem to apply when the space bound is less strict, i.e., super-exponential. Do these kinds of P systems with active membranes also exhibit the same computing power as Turing machines working under the same space constraints?

It might also be interesting to analyze the behavior of families of P systems with active membranes working in logarithmic space. In this case, there are two complications. First of all, we must slightly change the notion of space complexity, in order to allow for a "read-only" input multiset that is not counted when the space required by the P system is measured (similarly to the input tape of a logspace Turing machine). Furthermore, the notion of uniformity used to define the families of P systems should be weakened, since polynomial-time Turing machines constructing the families might be able to solve the problems altogether by themselves. More general forms of uniformity have already been investigated [2], and that work is going to be useful when attacking this problem.

References

1. J.T. Gill: Computational complexity of probabilistic Turing machines. *Proc. Sixth Annual ACM Symposium on Theory of Computing, STOC '74*, 1974, 91–95.
2. N. Murphy, D. Woods: The computational power of membrane systems under tight uniformity conditions. *Natural Computing*, 10 (2011), 613–632.
3. M.J. Pérez-Jiménez: A computational complexity theory in membrane computing. *Membrane Computing, 10th International Workshop, WMC 2009* (Gh. Păun et al., eds.), LNCS 5957, Springer, 2010, 125–148.
4. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Elementary active membranes have the power of counting. *International Journal of Natural Computing Research*, 2 (2011), 35–48.
5. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Introducing a space complexity measure for P systems. *International Journal of Computers, Communications & Control*, 4 (2009), 301–310.
6. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: P systems simulating oracle computations. *Membrane Computing, 12th International Conference, CMC 2011* (M. Gheorghe et al., eds.), LNCS 7184, Springer, 2012, 346–358.
7. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science*, 22 (2011), 65–73.
8. Gh. Păun: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6 (2011), 75–90.
9. P. Sosík, A. Rodríguez-Patón: Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73 (2007), 137–152.
10. S. Toda: PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20 (1991), 865–877.

10 The P-Conjecture and Hierarchies

Niall Murphy

Facultad de Informática
Universidad Politécnica de Madrid, Madrid, Spain
niall.murphy@upm.es

The P conjecture deals with the power of polarizations associated with the membranes of a P system with active membranes (looking for the borderline between efficiency and non-efficiency in this framework).

Required Notions: active membranes (without charges), weak non-elementary membrane division, elementary membrane division, dissolution rules, recognizer P systems, uniform families

Conjecture 1 (The P-conjecture (Problem F in [8])). The class of all decision problems solvable in polynomial time by active membranes without charges, using evolution, communication, dissolution, and division rules for elementary membranes, is equal to **P**.

Attempting to resolve the P-conjecture and its restrictions [1, 3, 4, 5, 9, 10] has resulted in many interesting new techniques, such as dependency graphs [3], for proving upper-bounds on membrane systems. Hopefully solving the P-conjecture will need new tools that yield deep revelations and open new questions into the nature of P-systems.

If the P-conjecture is proved to be true, then membrane systems with elementary division rules characterize **P** while those with weak non-elementary division rules characterize **PSPACE** [1, 9]. The deterministic class **P** is also the 0th level of Polynomial Hierarchy [6]. The complete problems of each successive level of the hierarchy seem to require increasing interleaving of nondeterminism and co-nondeterminism. The union of all levels is referred to as **PH** which is contained in **PSPACE**.

Characterizing each level of the Polynomial Hierarchy with a single model might give us clues to the role of nondeterminism in P systems. For example, could it be that division of different numbers of nested membranes is the membrane computing equivalent of alternating universal and existential nondeterminism?

Conjecture 2. Uniform families of active membrane systems using weak non-elementary division, without charges, and with a membrane structure of depth $d + 1$ can solve exactly those problems complete for the dth level of the Polynomial Hierarchy.

Some ideas for showing the lower-bound of this conjecture can be found in [7].

Continuing with the idea of hierarchies, a characterization of each level of the **NC** (or **AC**) hierarchy [6] may shed new light on the role of parallelism in membrane systems. The **NC** hierarchy represents a spectrum of problems ranging from constant time, to parallel logarithmic time, up to and (it is conjectured) not including the seemingly inherently sequential **P** [2]. A good place to start learning more about the factors that limit and permit parallelism in P systems might be a membrane characterization of the $\mathbf{NC} \overset{?}{=} \mathbf{P}$ problem (the so called "frontier of parallelism").

Problem 3. Is it possible to parameterize a resource or rule of a membrane computing model such that when the parameter is i it characterizes the ith level of the **NC** or **AC** hierarchy.

References

1. A. Alhazov, M.J. Pérez-Jiménez: Uniform solution to QSAT using polarizationless active membranes. *Machines, Computations and Universality (MCU)* (J. Durand-Lose, M. Margenstern, eds.), LNCS 4664, Springer, 2007, 122–133.
2. R. Greenlaw, H. James Hoover, W.L. Ruzzo: *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.
3. 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 (2006), 593–611.
4. N. Murphy, D. Woods: The computational power of membrane systems under tight uniformity conditions. *Natural Computing*, 10 (2011), 613–632.
5. N. Murphy, D. Woods: Active membrane systems without charges and using only symmetric elementary division characterise P. *Membrane Computing, 8th International Workshop, WMC 2007*, LNCS 4860, Springer, 2007, 367–384.
6. C.H. Papadimitriou: *Computational Complexity.* Addison Wesley, 1993.
7. A.E. Porreca, N. Murphy: First steps towards linking membrane depth and the Polynomial Hierarchy. *Proc. 8th Brainstorming Week on Membrane Computing*, Sevilla, 2010, 255–266.
8. Gh. Păun: Further twenty six open problems in membrane computing. *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla (Spain), 2005, 249–262.
9. P. Sosík, A Rodríguez-Patón: Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73 (2007), 137–152.
10. D. Woods, N. Murphy, M.J. Pérez-Jiménez, A. Riscos-Núñez: Membrane dissolution and division in P. *Unconventional Computation*, 5715 (2009), 262–276.

11 Seeking Sharper Frontiers of Efficiency in Tissue P Systems

Mario J. Pérez-Jiménez[1], **Agustín Riscos-Núñez**[1], **Miquel Rius-Font**[2], **Álvaro Romero-Jiménez**[1]

[1] Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
marper@us.es

[2] Department of Applied Mathematics IV
Universitat Politécnica de Catalunya, Casteldefels, Spain
mrius@ma4.upc.edu

In a P system, there are several ingredients which concur to their efficiency; varying them, one can get efficient systems (able to solve computationally hard problems in polynomial time) or non-efficient systems (e.g., solving **NP**-hard problems in an exponential time). The borderline between efficiency and non-efficiency is thus a problem of a central interest. This issue is explored here for tissue P systems, where the respective research started later than for cell P systems.

Required Notions: tissue P systems, complexity classes, cell division, cell separation, symport/antiport rule

A *tissue P system with symport/antiport rules* $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$, of degree $q \geq 1$ can be viewed as a set of q cells, labeled by $1, \dots, q$, with an environment labeled by 0 which initially have an arbitrary number of copies of some kind of objects, and a set of rules which can be of several types: communication, division or separation (see [3, 4] for details).

For each natural number $k \geq 1$, $\mathbf{TDC}(k)$ (respectively, $\mathbf{TDS}(k)$ or $\mathbf{TDA}(k)$) is the class of recognizer tissue P systems with cell division and communication rules (allowing only **s**ymport or **a**ntiport rules, respectively) of length at most k. Similarly, by considering separation rules instead of division rules, we denote $\mathbf{TSC}(k)$, $\mathbf{TSS}(k)$ and $\mathbf{TSA}(k)$ respectively. We denote by $\mathbf{PMC_R}$ the set of all decision problems which can be solved in a uniform way and polynomial time by means of families of systems from a class $\mathbf{R}$ of recognizer tissue P systems.

(A) Tissue P systems with cell division and with cell separation

By using the dependency graph technique, it has been proved that $\mathbf{P} = \mathbf{PMC}_{TDC(1)} = \mathbf{PMC}_{TSC(1)}$ [2, 3]. Furthermore, efficient and uniform solutions to the SAT problem by using systems from $\mathbf{TDC}(3)$ [1] and from $\mathbf{TSC}(8)$ [3] have been given. Recently, the last result has been improved to $\mathtt{SAT} \in \mathbf{PMC}_{TSC(3)}$ [6].

Problem 1. Assuming $\mathbf{P} \neq \mathbf{NP}$, in the framework of tissue P systems with cell division/cell separation, a frontier of the tractability is obtained when passing from communication rules with length 1 to communication rules with length at most 3. Does passing from 1 to 2, amounts to passing from non–efficiency to efficiency?

Conjecture: $\mathbf{NP} \cup \mathbf{co}\text{-}bfNP \subseteq \mathbf{PMC}_{TDC(2)}$.

(B) The role of direction in communication rules

Next, we deal with complexity aspects of tissue P systems with cell divison/celll separation where only symport or antiport rules are allowed. We have: $\mathbf{P} = \mathbf{PMC}_{TDA(1)} = \mathbf{PMC}_{TSA(1)}$, and $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{TDA(3)} \cap \mathbf{PMC}_{TSA(3)}$. Thus, assuming $\mathbf{P} \neq \mathbf{NP}$, a first frontier between efficiency and non-efficiency is obtained in the above framework when passing from communication rules with length 1 to communication rules with length at most 3.

Problem 2. What about the complexity classes $\mathbf{PMC}_{TDA(2)}$, $\mathbf{PMC}_{TSA(2)}$, $\mathbf{PMC}_{TDS(k)}$ and $\mathbf{PMC}_{TSS(k)}$, for all $k \geq 1$?

Conjecture: $\mathbf{P} = \mathbf{PMC}_{TSA(2)}$, and for all $k \geq 1$, $\mathbf{P} = \mathbf{PMC}_{TSS(k)}$.

If this conjecture is true, then passing from symport rules to antiport rules with length at least three, amounts to passing from non–efficiency to efficiency, in the framework of tissue P systems with cell separation.

(C) The role of the environment

Classical tissue P systems have a special alphabet associated with the environment, whose elements appear at the initial configuration of the system, in an arbitrary large amount of copies. What may happen if this property is removed, that is, if the alphabet associated to the environment were empty? We use a "hat" to indicate the case when the environment is initially empty.

Recently, have been proved that, for each $k \geq 1$, $\mathbf{PMC}_{\mathcal{TDC}(k)} = \mathbf{PMC}_{\widehat{\mathcal{TDC}}(k)}$ [5], that is, in the framework of tissue P systems with cell division the role of the environment is not relevant from the complexity point of view.

Conjecture: For each $k \geq 1$, $\mathbf{P} = \mathbf{PMC}_{\widehat{\mathcal{TSC}}(k)}$.

If this conjecture is true, then in the framework of tissue P systems with cell communication the following holds: (a) passing from separation rules to division rules (length at least three) amounts to passing from non–efficiency to efficiency; and (b) the environment provides a new borderline of efficiency.

References

1. D. Díaz, M.A. Gutiérrez, M.J. Pérez-Jiménez, A. Riscos-Núñez: A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, 404 (2008), 76–87.
2. R. Gutiérrez-Escudero, M.J. Pérez-Jiménez, M. Rius-Font: Characterizing tractability by tissue-like P systems. *Membrane Computing. 10th International Workshop, WMC 2009, Curtea de Argeş, August 2009. Revised Selected and Invited Papers*, LNCS 5957, Springer, Berlin, 2010, 289–300.
3. L. Pan, M.J. Pérez-Jiménez: Computational complexity of tissue–like P systems. *Journal of Complexity*, 26 (2010), 296–315.

4. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez: Tissue P systems with cell division. *International Journal of Computers, Communications & Control*, 3 (2008), 295–303.
5. M.J. Pérez-Jiménez: The role of the environment in tissue P systems with cell division. Submitted, 2012.
6. M.J.Pérez-Jiménez, P. Sosík: Improving the efficiency of tissue P systems with cell separation. Submitted, 2012.

12 Time-Free Solutions to Hard Computational Problems

Matteo Cavaliere

National Center for Biotechnology, CNB - CSIC, Madrid, Spain
mcavaliere@cnb.csic.es

P systems are usually synchronized, a unique clock marks the time for all components, and in each time unit each component evolves (usually, in the maximal parallel manner). In time-free (and clock-free) systems, this strong assumption is removed. Up to now, the efficiency of P systems was not investigated also for this case.

Required Notions: Time-free P system, synchronization, recognizing P system, uniform/semi-uniform solution.

12.1 Motivations

Living cells have division rates that are highly heterogeneous (even in identical environmental conditions), consequence of their stochastic gene expression, [1]. Therefore, the possibility of programming living cells should not assume the presence of uniform replication rates. Ideally, one should construct “cellular computers” whose functioning is independent of cellular division rates. We suggest that such problem can be addressed in the framework of membrane computing by extending the notion of time-freeness ([4]) to the idea of semi-uniform solutions of computational problems based on membrane divisions ([3]).

12.2 Timed Recognizer P Systems

From [4] we recall the notion of timed P system.

A *timed P system* $\Pi(e)$ can be constructed by adding to a (standard) P system Π a time-mapping $e : R \longrightarrow \mathbb{N}$, where R is the set of rules of Π. The time-mapping specifies the *execution times* for the rules.

A timed P system $\Pi(e)$ works in the following way. We suppose to have an external clock that marks time-units of equal length (called *steps*), starting from step 0, when the system is present in its *initial configuration*.

At each step, all the rules that can be started, in each region, and for each membrane have to be started (maximal parallel and nondeterministic use of rules). *When a rule r is started at step j, then its execution terminates (the rule is completed) at step $j + e(r)$, that means the rule lasts $e(r)$ steps. The objects and the membranes produced by the rule are available – can be subject of other rules – only starting from the step $j+e(r)+1$.* When a rule r is started, then the occurrences of symbol-objects and the membrane subject by this rule cannot be anymore subject of other rules.

A computation *halts* when no rule can be started in any region and there are no rules in execution (such configuration is called *halting*). We say that the computation halts in k steps, if the external clock marks step k when the last rules of the computations are completed.

From [3] we recall the notion of recognizer P systems. A *decision problem* X is a pair (I_X, Θ_X) where I_X is a countable language over a finite alphabet (the elements are called instances), and Θ_X is a predicate (a total boolean function) over I_X.

A *recognizer P system* is a P system such that: (*i*) the working alphabet contains two distinguished elements *yes* and *no*; (*ii*) all computations halt; and (*iii*) if $\mathcal{C}$ is a computation of the system, then either object *yes* or object *no* (but not both) must have been released into the environment, and only when the last rules of the computation have been completed.

We extend recognizer P systems by proposing the following timed variant: a *recognizer timed P system* is a *timed P system* with properties (i), (ii), (iii) above.

In recognizer timed P systems, we say that a computation is an *accepting computation* (respectively, *rejecting computation*) if the object *yes* (respectively, *no*) appears in the environment associated with the corresponding halting configuration.

12.3 Time-Free Solutions to Decision Problems

Let $X = (I_X, \Theta_X)$ be a decision problem. Let $\Pi = \Pi_u, u \in I_X$, a (countable) family of recognizer P systems.

We say that the family Π is *sound* (with respect to X) if for each instance of the problem $u \in I_X$ such that there exists an accepting computation of Π_u, we have $\Theta_X(u) = 1$.

We say that the family Π is *complete* (with respect to X) if for each instance of the problem $u \in I_X$ such that $\Theta_X(u) = 1$, every computation of Π_u is an accepting computation.

We say that the family Π is *polynomially bounded* if there exists a polynomial function $p(n)$ such that, for each $u \in I_X$, all computations in Π_u halts in, at most, $p(|u|)$ steps.

We can now formalize the original motivations: A solution to a problem is time-free if its soundness, its completeness and its polynomial bound do not depend on the time of execution associated to the rules of the constructed systems.

We say that the family Π is *time-free sound* (with respect to X) if, for any time-mapping e, the family $\Pi_e = \Pi_u(e), u \in I_X$, is sound with respect to X.

We say that the family Π is *time-free complete* (with respect to X) if, for any time-mapping e, the family $\Pi_e = \Pi_u(e), u \in I_X$, is complete with respect to X.

We say that the family Π is *time-free polynomially bounded* if, for any time-mapping e, the family $\Pi_e = \Pi_u(e), u \in I_X$, is polynomially bounded.

We can now adapt the definition of semi-uniform solutions, as given in [3], and consider time-free semi-uniform solutions.

Let $X = (I_X, \Theta_X)$ a decision problem. We say that X is solvable in a *time-free polynomial time* by a family of recognizer P systems $\Pi = \Pi_u, u \in I_X$, if the following are true:

- the family Π is polynomially uniform by a Turing machine; that is, there exists a deterministic Turing machine working in polynomial time which constructs the system Π_u from the instance $u \in I_X$.
- the family Π is time-free polynomially bounded.
- the family Π is time-free sound and time-free complete (with respect to X).

We say that the family Π is a *time-free semi-uniform solution* to the decision problem X.

In other words, to provide a time-free solution one must construct the family of systems Π in polynomial time (sequential time by deterministic Turing machines) and the constructed family must be "fast" (polynomially bounded), sound and complete with respect to the considered problem X, and these properties must be independent of the execution time of the rules (i.e., they must be fulfilled independently of the time-mapping considered).

The definition of time-free semi-uniform solution captures the problem informally discussed in the Motivations. The basic *question* consists in finding a class of membrane systems for which it is possible to construct time-free semi-uniform solutions to hard computational problems. The simplest possibility is to transform the solutions already present in literature into time-free solutions (e.g., could the solution given in [2] be adapted to become a time-free solution?).

Another interesting problem is to find classes of membrane systems that are powerful enough to solve complex problems, but simple enough to allow an automatic (i.e., algorithmic) checking of their time-freeness.

References

1. M.B. Elowitz, J. Levine, E.D. Siggia, P.S. Swain: Stochastic gene expression in a single cell. *Science*, 297 (2002), 5584.
2. Gh. Păun: P systems with active membranes: Attacking NP complete problems. *Journal of Automata, Language and Combinatorics*, 6 (2001), 75–90.
3. M.J. Pérez-Jiménez, A. Riscos-Núñez, A. Romero–Jiménez, D. Woods: Complexity – membrane division, membrane creation. Chapter 12 in [5], 302–336.

4. M. Cavaliere, D. Sburlan: Time-independent P systems. *Membrane Computing. Int. Workshop WMC 2004*, Milan, Italy, 2004 (G. Mauri et al., eds.), LNCS 3365, Springer, 2005, 239–258.
5. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing.* Oxford Univ. Press, 2010.

13 Fypercomputations

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
Bucharest, Romania, and

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
gpaun@us.es

Following the model of *hypercomputation* (computing beyond the "Turing barrier"), we propose here the term *fypercomputation* to name the research area of "solving polynomially problems which are (at least) **NP**-complete". Some ideas from/for MC are mentioned.

Required Notions: membrane division, membrane creation, hypercomputing, SN P system, reaction system, accelerated P system

Looking for ideas which would lead to computing devices able to compute "beyond the Turing barrier" is already a well established research area of computing theory; such devices are said to be able of doing *hypercomputations.* It is also a dream and a concern of computability to speed-up computing devices; a name was proposed in [7] (the idea was further elaborated in [8]) for the case when this leads to polynomial solutions to problems known to be (at least) **NP**-complete: *fypercomputing* – with the initial **F** coming from "fast".

In short: *fypercomputing means going polynomially beyond* **NP**.

The model we have in mind is that of hypercomputations, already with a large literature (we only mention the recent survey from [10]). More than a dozen of ideas were proposed and proved to reach the goal of computing "beyond Turing": oracles (already considered by Turing), introducing real numbers in the device, accelerating the functioning of machines, using ingredients of an analogical nature and so on. Many of these ideas can probably lead not only to hypercomputations, but also to fypercomputations, both in MC and in other frameworks.

Although not clustered under a good name, such as hypercomputation (there are periodical meetings dedicated to this research direction), there also are many papers which can be placed under the flag of fypercomputing. They exploit ideas from physics, such as [9], propose analogical computations, such as [1]. Also the area of DNA computing is full of such ideas.

The literature of membrane computing abounds in papers dealing with fypercomputations. In most cases, polynomial solutions to **NP**-complete problems – often, also of **PSPACE**-complete problems – are obtained, by making use of a space-time trade-off, with the space obtained during the computation, by means of operations inspired from biology. The most investigated operations of this kind are *membrane division* (with variants: separation, budding, etc.) and *membrane creation*.

Further two similar ideas were also explored. The first one is based on string replication (see [3] for details), the second one is that of considering arbitrarily large *pre-computed resources* (see, e.g., [6]), but the last idea is only briefly investigated so far. Issues related to the conditions to be imposed to the given pre-computed resources should be further considered.

Three more ideas, essentially different from the previous ones, were proposed in [8] and need additional research efforts.

(1) The first candidate is the *acceleration*, an old one in computer science: a "clever" computing device learns from its own functioning; after performing a step in a time unit, it performs better for the second step, which is completed in half of the time necessary for the first step – and so on, at each step halving the time with respect to the previous step. If the first step takes one time unit, then the second one takes 1/2 time units, the third one 1/4 and so on, hence in two time units the computation ends.

Important: we have here two clocks, an internal one, of the machine, and an external one, of the observer. The internal clock is faster and faster, so that the computation ends in two time units *measured by the external clock*, that of the observer/user.

Accelerated Turing machines can solve the halting problem, hence they compute what usual Turing machines cannot. See references in [2], where the idea is extended to P systems: starting from the biological observation that "smaller is faster" and using membrane creation rules to create "faster reactors" (inner membranes), in an unbounded hierarchy, one can obtain P systems which "compute the uncomputable".

This trick can be used also in complexity, but we have to be cautious: we accelerate in order to get a speed-up... In two (external) time units we solve any problem, whatever complex it is. A way to make the things interesting is to accelerate only parts of a P system, thus having several levels of time speed. For instance, we can accelerate only (i) some elementary membranes, or (ii) only some rules (a given rule takes one time unit for the first application, half for the second application, and so on), or (ii) to have "accelerated objects" (the descendants of an object react faster than the father object, irrespective which are the rules which act on them and irrespective of the membranes where they are). Precise definitions should be found and their usefulness explored.

(2) The previous ideas suggest the following speculation. We mentioned that we have (at least) two clocks, an external one, of the observer (or of the higher membranes in the structure) and the local clock(s), of the accelerated element, membrane, rule, object. Always, the inner clock is (much) faster than the external one, it performs sometimes an exponential number of steps while the external one only ticks once. We can then imagine that the inner time is orthogonal to the external time, hence the time has a 2D structure: the observer only senses one dimension of time, but certain "processors" can run along the other dimension, doing computations at-no-time for the observer. This looks very much as using oracles. Again, good definitions have to be found and explored.

(3) One further idea, proved in [8] to lead to fypercomputations comes from the recently introduced *reaction systems* (we call them *R systems*) – see [4], [5]. One of the crucial postulates of R systems concerns the fact that one works with ω multisets: an object either is not present, or it is present in arbitrarily many copies. This assumption can be extended also to P systems. More exactly, we consider P systems which contain certain distinguished elementary membranes, whose objects are present in arbitrarily many copies (for instance, if an object a is introduced from outside in such a membrane, then inside the membrane it immediately becomes a^ω). In [8], such a system is called ωP system and it is proved that `SAT` can be solved (in a uniform way) in a polynomial time by an ωP system.

The construction in [8] uses cooperative rules; we do not know whether the result can be improved by imposing the restriction to use only non-cooperative rules.

References

1. J.J. Arulanandham, C.S. Calude, M.J. Dinneen: Balance machines: computing = balancing. *Aspects of Molecular Computing, 2004*, LNCS 2950, Springer, 2004, 148–161.
2. C.S. Calude, Gh. Păun: Bio-steps beyond Turing. *BioSystems*, 77 (2004), 175–194.
3. J. Castellanos, Gh. Păun, A. Rodriguez-Patón: Computing with membranes: P systems with worm-objects. *Proc. IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000*, La Coruna, Spain, 2000, 64–74.
4. A. Ehrenfeucht, G. Rozenberg: Basic notions of reaction systems, *Proc. DLT 2004*, LNCS 3340, Springer, 2004, 27–29.
5. A. Ehrenfeucht, G. Rozenberg: Reaction systems. *Fundamenta Informaticae*, 75 (2007), 263–280.
6. T.-O. Ishdorj, A. Leporati: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing*, 7 (2008), 519–534.
7. Gh. Păun: Membrane computing at twelve years. Back to Turku. *Proc. UC 2011*, Turku, Finland, June 2011, LNCS 6714, Springer, 2011, 36–37.
8. Gh. Păun: Towards "fypercomputations" (in membrane computing), LNCS 6714, Springer, 2011, 36–37.

9. V. Putz, K. Svozil: Can a computer be "pushed" to perform faster-than-light? *Proc. UC10 Hypercomputation Workshop "HyperNet 10"*, Univ. of Tokyo, June 22, 2010.
10. A. Syropoulos: *Hypercomputation: Computing Beyond the Church-Turing Barrier.* Springer, 2008.

14 Numerical P Systems

Cristian Ioan Vasile[1], **Ana Brânduşa Pavel**[1], **Ioan Dumitrache**[1], **Gheorghe Păun**[2]

[1]Department of Automatic Control and Systems Engineering
Politehnica University of Bucharest, Romania
{cvasile, apavel, idumitrache}@ics.pub.ro

[2]Institute of Mathematics of the Romanian Academy
Bucharest, Romania, and

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
gpaun@us.es

Extensions of numerical P systems motivated by using such systems in robot controlling are mentioned and problems occurring in this framework are formulated.

Required Notions: numerical P system, complexity, promoters-inhibitors, catalyst

Numerical P systems are a class of computing models (introduced in [5]; see also Chapter 23.6 of [6]) inspired both from the cell structure and economics: numerical variables evolve in the compartments of a cell-like structure by means of so-called *production–repartition programs*. The variables have a given initial value and the production function is usually a polynomial, whose value for the current values of variables is distributed among variables in the neighboring compartments according to the "repartition protocol". In this way, the values of variables evolve; all positive values taken by a specified variable are said to be computed by the P system.

These systems were recently used in a series of papers (see references in [1]) for implementing controllers for mobile robots; in this framework the P systems work in the computing mode: an input is introduced in the form of the values of some variables and an output is produced, as the value of other variables. Furthermore, in the robot control context, the so-called *enzymatic* numerical P systems were introduced and used, [2], [3], [4]. Such systems correspond to *catalytic P systems* in the “general” membrane computing: a program is applied only if the value of the associated enzyme is strictly greater than the smallest value of any variable involved in the production polynomial. Enzyme variables are not consumed or produced by the rules which they catalyze, but can be changed by the rules for which they do not act as catalysts. Therefore, their values can evolve during the computational process.

Tissue numerical P systems are also considered in [8], with parallel use of programs. If in each membrane, at each step, we use a maximal set of programs (programs are selected nondeterministically, and a set of programs is applied only if it is maximal, no further program can be added to it in such a way that the new set is still applicable). Two possibilities appear: (i) a variable can appear only in *one* production function, and this is the only restriction in choosing (nondeterministically) the programs to apply in a step, and (ii) if two or more programs which are enabled at a computation step, i.e., they satisfy the condition imposed by the associated enzymes, share variables in their production functions, then they will all use the current values of those variables (we denote this with *allP*).

A large variety of classes of numerical P systems appears in this way: (1) enzymatic or non-enzymatic, (2) deterministic or nondeterministic, (3) sequential, all-parallel, one-parallel, (4) used in the generating, computing, accepting mode; further variants can be added. By combining all these, a plethora of classes of numerical P systems appear.

We do not recall here the definition of numerical P systems, with or without enzyme control, but we refer the reader to the papers mentioned above.

We only mention that the family of sets of numbers $N^+(\Pi)$ computed by numerical P systems Π with at most m membranes, production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial, is denoted by $N^+P_m(poly^n(r), seq),\ m \geq 1, n \geq 0, r \geq 0$, where the fact that we work in the sequential mode (in each step, only one program is applied) is indicated by *seq*. If one of the parameters m, n, r is not bounded, then it is replaced by $*$. (Both in $N^+(\Pi)$ and in $N^+P_m(poly^n(r), seq)$, the superscript $+$ indicates the fact that as the result of a computation we only consider positive natural numbers, zero excluded. If any value is accepted, then we remove the superscript $+$.) When tissue systems are used, we write $NtP_m(poly^n(r), \alpha, \beta)$.

Here are a few results from [5] and [8].

Theorem 1. $NRE = N^+P_8(poly^5(5), seq) = N^+P_7(poly^5(6), seq) = NP_7(poly^5(5), enz, seq) = NtP_*(poly^1(11), enz, oneP) = NP_{254}(poly^2(253), enz, allP, det)$.

Whether or not the parameters appearing in these results are optimal or not is an open problem.

Only a few of the many cases mentioned above were so far investigated, the other ones wait for research efforts.

In particular, we have seen that enzymes improve the universality results in terms of the complexity of used polynomials, both in the cell-like case and the tissue-like case, provided that the evolution programs are used in a parallel manner. However, two different types of parallelism were used in the two cases; can the one-parallel mode (used for tissue P systems) be used also in the cell-like case?

Similar extensions of "general" notions in membrane computing to numerical P systems remain to be examined, and this is a rich research topic. For instance, other ways of using the programs can be considered: minimally parallel, with bounded parallelism, asynchronously. Then, we can also consider rules for handling membranes, such as membrane division and membrane creation. These operations are the basic tools by which polynomial solutions to computationally hard problems, typically, **NP**-complete problems, are obtained in the framework of P systems with symbol objects. Is this possible also for numerical P systems?

A current issue in membrane computing is to find classes of P systems which are not universal. This extends also to numerical P systems.

Of course, a natural research topic is to further explore the use of numerical P systems in controlling robots. In this framework, an important question is to develop a complexity theory based on numerical P systems: define specific complexity classes, compare them with existing classes, look for ways to speed-up computations (see also the previous suggestion, to bring to numerical P systems further ideas investigated for symbol object P systems, in particular, tools to create an exponential working space in polynomial time).

And so on and so forth, a wealth of research ideas, which supports our belief that numerical P systems deserve further research efforts.

References

1. C. Buiu, C.I. Vasile, O. Arsene: Development of membrane controllers for mobile robots. *Information Sciences*, 187 (2012), 33–51.
2. A.B. Pavel, C. Buiu: Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, in press, DOI: 10.1007/s11047-011-9286-5, 2011
3. A.B. Pavel, O. Arsene, C. Buiu: Enzymatic numerical P systems – a new class of membrane computing systems. *The IEEE Fifth Intern. Conf. on Bio-Inspired Computing. Theory and applications. BIC-TA 2010*, Liverpool, Sept. 2010, 1331–1336.
4. A.B. Pavel, C.I. Vasile, I. Dumitrache: Robot localization implemented with enzymatic numerical P systems. Submitted, 2012
5. Gh. Păun, R. Păun: Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, 73 (2006), 213–227.
6. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford Univ. Press, 2010.
7. The P Systems Web Page: `http://ppage.psystems.eu`.
8. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: On the power of enzymatic numerical P systems. Submitted, 2011.

15 P Systems Formal Verification and Testing

Florentin Ipate[1], **Marian Gheorghe**[1,2]

[1]Department of Computer Science, University of Piteşti
Piteşti, Romania
florentin.ipate@ifsoft.ro

[2]Department of Computer Science, University of Sheffield
Sheffield, UK
m.gheorghe@dcs.shef.ac.uk

Issues related to formal verification (through model checking) and model-based testing of various classes of P systems are mentioned – especially,extensions of existing results are pointed out, either using new tools or tackling classes of P systems useful in various frameworks.

Required Notions: cell P system, active membrane, tissue P system, model checking, model based testing, P-lingua

In the last years, more complex applications of P systems have been built and used to study the behaviour of various systems in biology, economics and linguistics. Models based on P systems have been introduced to investigate problems in distributed computing and process synchronisation. All these applications and models allow to simulate the systems studied and to identify their various properties. It is important that all these applications are correctly implemented and produce the right results. In order to check their correctness, formal verification and testing methods and tools are employed. Formal verification based on model checking and model-based testing will be presented below.

15.1 Formal verification of P systems through model checking

Model checking is an automated technique for verifying if a state-based model of a system meets a given specification. Using a temporal logic formula searches through the entire state space to check whether the property holds or fails are executed. If a property violation is discovered, then a counterexample is returned [3]. Formal verification of P systems using model checking has attracted a significant amount of research in recent years, using tools such as Maude [1], PRISM [2], NuSMV [8], Spin [9] or ProB [7]. The decidability of model checking properties for P systems has also been studied in [4]

Most research has focussed on cell P systems with a static structure, but, more recently, P systems with active membranes, in particular with division rules, have also been investigated [10]. This is a significant advance from a practical point of view since P systems with division rules are commonly used to devise efficient solutions to computationally hard problems. However, the state explosion problem normally associated with model checking still hampers such approaches, in particular in the case of P systems with cell division, for which the number of states can grow exponentially. To address this, more efficient implementations, as well as ways of reducing the number of states through the construction of *approximate* state-based models (possibly using inference techniques such as that presented in [11]) might be investigated.

The following problems regarding the formal verification of P systems are expected to be investigated:

- new methods and techniques for formally verifying variants of P systems with a dynamical structure used to model systems solving problems from computer science or engineering;
- identification of invariants using Daikon, a tool which dynamically detects program properties based on execution traces;
- developing an integrated environment for specifying and formally verifying P systems using P-lingua, Daikon, and one or more model checking tools;
- extending the existing approaches to other classes of P systems (e.g., tissue P systems).

15.2 Model-based testing of P systems

Testing is the main means of software validation and an essential part of system development; all software applications, irrespective of their use and purpose, are tested before being released. In testing, programs are run on a set of sample data in order to expose faults in the code. This means that an essential (and in many cases the most time consuming) part of testing is selecting such sample data. This process is called *test generation.* As in the last years there have been significant developments in using the P systems paradigm to model, simulate and formally verify various systems (in biology, economics, linguistics, graphics, computer science, etc.), test generation methods for systems modelled as P systems must also exist.

In the last years, a number of approaches to testing P systems have been developed. One approach involves the definition of a number of coverage criteria (such as simple rule coverage, in which each rule of the P system must be covered at least once, and more complex variants) and the selection of test data to meet these criteria [5]. An extension of this strategy involves mutation analysis: here, a test selection criterion is defined by producing a slightly modified version of the system (called a mutant) and the selected test data must distinguish between the original model and the mutant [8]. Another approach to P system testing is based on finite state machine conformance techniques [6]. This involves the construction of a state-based approximation of the P system (called a deterministic finite cover automaton) and the application of conformance testing techniques for such a finite state model [12].

Essentially, all the aforementioned techniques have been developed in the context of cell P systems with a fixed structure. The challenge for the future is to extend these to P systems with active membranes, as well as to other types of membrane systems. In particular, the development of a testing approach for tissue P systems, for which the interaction with the environment is conceptually close to the input/output behaviour of interactive systems, is expected, and may have an important practical impact. Ultimately, suitable tools will have to be developed and integrated within the aforementioned modeling, verification and testing.

Acknowledgement. This work was partially supported by project MuVet, Romanian National Authority for Scientific Research (CNCS UEFISCDI) grant number PN-II-ID-PCE-2011-3-0688.

References

1. O. Andrei, G. Ciobanu, D. Lucanu: A rewriting logic framework for operational semantics of membrane systems. *Theoretical Computer Science*, 373 (2007), 163–181.
2. F. Bernardini, M. Gheorghe, F.J. Romero-Campero, N. Walkinshaw: A hybrid approach to modelling biological systems. *Int. Workshop on Membrane Computing*, WMC 2007, LNCS 4860, Springer, 2007, 138–159.
3. E.M. Clarke, O. Grumberg, D.A. Peled: *Model checking.* MIT Press, Cambridge, MA, USA, 1999.
4. Z. Dang, O.H. Ibarra, C. Li, G. Xie: On the decidability of model-checking for P systems. *Journal of Automata, Languages and Combinatorics* 11 (2006), 279–298.
5. M. Gheorghe, F. Ipate, C. Dragomir: Formal verification and testing based on P Systems. *Int. Workshop on Membrane Computing*, WMC 2009, LNCS 5957, Springer, 2010, 54–65.
6. F. Ipate, M. Gheorghe: Finite state based testing of P systems. *Natural Computing*, 8 (2009), 833–846.
7. F. Ipate, A. Ţurcanu: Modelling, verification and testing of P systems using Rodin and ProB. *Ninth Brainstorming Week on Membrane Computing*, 2011, 209–220.
8. F. Ipate, M. Gheorghe, R. Lefticaru: Test generation from P systems using model checking. *Journal of Logic and Algebraic Programming*, 79 (2010), 350–362.
9. F. Ipate, R. Lefticaru, C. Tudose: Formal verification of P systems using Spin. *International Journal of Foundations of Computer Science*, 22 (2011), 133–142.
10. F. Ipate, R. Lefticaru, I. Pérez-Hurtado, M.J. Pérez-Jiménez, C. Tudose: Formal verification of P systems with active membranes through model checking. *Int. Conference on Membrane Computing*, CMC12, Fontainebleau, 2011, 241–252.
11. F. Ipate: Learning finite cover automata from queries. *Journal of Computer and System Sciences*, 78 (2012), 221–244.
12. F. Ipate: Bounded sequence testing from deterministic finite state machines. *Theoret. Comput. Sci.*, 411 (2010), 1770–1784.

16 Causality, Semantics, Behavior

Oana Agrigoroaiei, Bogdan Aman, Gabriel Ciobanu

Institute of Computer Science, Romanian Academy, Iaşi Branch, Romania
gabriel@info.uaic.ro,oanaag@iit.tuiasi.ro

The connection of MC with process algebra is not very much explored, although this is a very promising research direction. Some issue related to causality, behavior equivalence, type systems, relationships with the Chemical Abstract Machine are mentioned here.

Required Notions: cell P system, maximal parallelism, synchronization, semantics, event, (bi)simulation

16.1 Causality

Consider standard transition P systems with promoters and inhibitors and dissolution; they can be described, up to simulating one transition step with several others, by transition P systems with just one membrane (and with promoters and inhibitors).

In [3] we have defined causality at both specific and general level for transition P systems with one membrane and without any other ingredients; specific causality depends on a certain evolution step, while general causality takes into consideration all possible evolution steps. Two questions arise immediately: whether this construction is extendible to P systems involving promoters and inhibitors and whether causality can be defined in a more static manner, without involving the membrane system. One of the results of [3] (Theorem 15) indicates that the latter problem is solvable by using the more *dynamic* notion of *general causality.*

A different problem related to causality concerns the relation between various forms of evolution in transition P systems: maximal parallelism, local maximal parallelism and unrestricted parallelism. How do causal relations change when we change the form of evolution for a given P system? We have works in progress concerning the relationship between maximal parallelism and unrestricted parallelism which we hope will also be useful in having a clearer image of what causality means for membrane systems. Moreover, we ask how are such causal relations connected with the object-based event structures we introduced in [2], where the focus was not on rule application but on the objects being produced. As always, we have to ask in what manner do results change if additional ingredients (especially promoters and inhibitors) are introduced.

Finally, the idea of "computing backwards" [1], which was also mentioned in [8], is strongly related to the notion of causality and it would be interesting to see how it can be used to clarify or even solve the problems proposed above.

16.2 Chemical Abstract Machine and P systems

The Chemical Abstract Machine (CHAM) [5] is suited to model asynchronous concurrent computations such as algebraic process calculi. Intuitively, the state of a system is like a chemical solution in which floating molecules can interact with each other according to reaction rules; a "magical" mechanism stirs the solution, allowing for possible contacts between molecules. In chemistry, this is the result of Brownian motion. The solution transformation process is obviously truly parallel: any number of reactions can be performed in parallel, provided that they involve disjoint sets of molecules.

The chemical abstract machine presents molecules in a systematic way as terms of algebras and refining the classification of rules. Some molecules do not exhibit interaction capabilities; those which are ready to interact are called ions. A solution can be heated to break complex molecules into smaller ones up to ions. Conversely, a solution can be chilled to rebuild heavy molecules from components. Furthermore, to deal with abstraction and hierarchical programming, a molecule is allowed to contain a sub-solution enclosed in a membrane, which can be somewhat porous to allow communication between the encapsulated solution and its environment. The chemical abstract machines all obey a simple set of structural laws. Each particular machine is given by adding a set of simple rules that specify how to produce new molecules from old ones. Unlike the inference rules classically used in structural operational semantics, the specific rules have no premises and are purely local.

Since P systems and CHAM start from the same premises, but use different notions, notations, and operational semantics and have different goals, it would be interesting to study the connections between these two fields.

16.3 Type Systems

Type theory is fundamental both in logic and computer science. Theory of types was introduced by B. Russell [9] in order to solve some contradictions of set theory. In computer science, type theory refers to the design, analysis and study of type systems. Generally, a type system is used to prevent the occurrences of errors during the evolution of a system. A type inference procedure determines the minimal requirements to accept a system or a component as well-typed.

P systems consider cells as mechanisms working in a maximal parallel and nondeterministic manner. However, the living cells do not work in such a way: a chemical reaction takes place only if certain quantitative constraints are fulfilled. In order to cope with such constraints, P systems should be enriched by adding a quantitative type discipline, and making use of type inference and principal typing [10]. We associate to each reduction rule a minimal set of constraints that must be satisfied in order to assure that by the application of this rule to a well-formed P system, we get a well-formed P system as well. A first step in this direction was done in [4] where a type system for P system with symport/antiport rules is given.

The type systems can be used in defining more general and simpler rules for P systems. For example, if $\mathbf{N_1}$ and $\mathbf{N_2}$ are some basic types, by considering a set of typed objects $V = \{X_1 : \mathbf{N_1}, X_2 : \mathbf{N_1}, X_3 : \mathbf{N_1}, A : \mathbf{N_2}\}$, the evolution rules of the form $X_i \rightarrow X_j$, $X_j \rightarrow A$, $1 \leq i \leq 3$, $1 \leq j \leq 3$, can be replaced by rules of a more general form:

1. $\mathbf{N_1} \rightarrow \mathbf{N_1}$ (any object of type $\mathbf{N_1}$ can evolve in any object of type $\mathbf{N_1}$);
2. $\mathbf{N_1} \rightarrow \mathbf{N_2}$ (any object of type $\mathbf{N_1}$ can evolve in any object of type $\mathbf{N_2}$).

16.4 Behavior Equivalence

Behavior equivalence is an important concept in biology needed for analyzing and comparing the organs behavior. For example, an artificial organ is the functional equivalent of the natural organ, meaning that both behave in a similar manner up to a given time; e.g. the artificial kidney has the same functional characteristics as an "in vivo" kidney. Recently, it is shown in [7] that the vas deferens' of the human, canine, and bull are equivalent in many ways, including histological similarities. In [6] are presented different methods for comparing protein structures in order to discover common patterns.

In membrane computing, two P systems are considered to be equivalent whenever they have the same input/output behavior. Such an equivalence does not take care of the evolution of the two systems. What does it mean that two P systems have equivalent (timed) behavior? Defining several equivalences, we offer flexibility in selecting the right one when verifying biological systems and comparing them. When a P system can be replaced in a context with another one such that the observed behavior is the same?

References

1. O. Agrigoroaiei, G. Ciobanu: Reversing computation in membrane systems. *Journal of Logic and Algebraic Programing*, 79 (2010), 278–288.
2. O. Agrigoroaiei, G. Ciobanu: Rule-based and object-based event structures for membrane systems. *Journal of Logic and Algebraic Programing*, 79 (2010), 295–303.
3. O. Agrigoroaiei, G. Ciobanu: Quantitative causality in membrane systems. *Proc. Twelfth International Conference on Membrane Computing*, Fontainebleau, 2011, 53–63.
4. B. Aman, G. Ciobanu: Typed membrane systems. *Int. Workshop on Membrane Computing*, WMC 2009, LNCS 5957, Springer, 2010, 169–181.
5. G. Berry, G. Boudol: The chemical abstract machine. *Theoretical Computer Science*, 96 (1992), 217–248.
6. I. Eidhammer, I. Jonassen, W. Taylor: Structure comparison and structure patterns. *Journal of Computational Biology*, 7 (2000), 685–716.
7. D.E. Leocadio, A.R. Kunselman, T. Cooper, J.H. Barrantes, J.C. Trussell: Anatomical and histological equivalence of the human, canine, and bull vas deferens. *The Canadian Journal of Urology*, 18 (2011), 5699–5704.

8. Gh. Păun: Some open problems collected during 7th BWMC. *Proc. Seventh Brainstorming Week on Membrane Computing*, 2009, vol. 2, 197–206.
9. B. Russell: *The Principles of Mathematics*, vol. I, Cambridge University Press, 1903.
10. J. Wells: The essence of principal typings. LNCS 2380, Springer, 2002, 913–925.

17 Kernel P Systems

Marian Gheorghe

Department of Computer Science, University of Piteşti, Romania

Department of Computer Science, University of Sheffield, UK
m.gheorghe@dcs.shef.ac.uk

The issue of a common generalization of several classes of P systems is proposed, and some basic ideas towards such a goal are presented.

Required Notions: tissue P system, P system with dynamic structure, regular expression

Different variants of P systems have been used for specifying simple algorithms [5, 2], classes of **NP**-complete problems [7] and various applications [6]. More specific classes of P systems have been recently considered for modeling some distributed algorithms and problems [8]. In many cases the evolution of the system investigated requires some specific behavior or the use of certain rules, maybe with some constraints, which are not always the same as the ones exhibited by the model in its initial definition. It helps in many cases to have some flexibility with the modeling approach, especially in the specification stage, as it shortens the model and makes it clearer. Although there is a powerful specification language, called P-lingua, with implementations for various variants of P systems [9], there is no unified framework that allows us to simulate, verify and test the behavior of the specified systems. In this respect, it is suggested here a *kernel P system* (*kP system*, for short) that, in the first stage, will be a low level specification language including the most used concepts from P systems.

The generic structure of a kP system might be a graph-like structure as in tissue P systems. Such a model uses a set of symbols, labels of membranes, rules of various types and a certain strategy to run them against the multiset of objects available in each region. The rules in each compartment will be of two types: (i) *object processing rules* which transform and transport objects between compartments or exchange objects between compartments and environment, and (ii) *system structure rules* responsible for changing the system's topology. Each rule has a guard, defined using activators and inhibitors in a general way. The execution strategy is defined such that maximally parallel or sequential manners are captured and each compartment has its own strategy. Rewriting and communication rules based on promoters and inhibitors are considered together with a special set of symport/antiport rules. Additional features like membrane division, creation, dissolution, bond creation and destruction are used to deal with the system structure.

The key concept of a compartment is first introduced and then the definition of a kP system.

Definition 1. *Given an alphabet A, of elements named* objects*, and an alphabet L of* labels*, a* compartment *is a tuple $C = (l, w_0, R^\sigma)$, where $l \in L$ is the label of the compartment, w_0 is the initial multiset over A, and R^σ denotes "the DNA code", i.e., the set of rules, denoted R, applied in this compartment and a regular expression, σ, over $Lab(R)$, the labels of the rules of R.*

Definition 2. *A* kernel P system *is a tuple $k\Pi = (A, L, IO, \mu, C_1, \ldots, C_n)$, where A and L are, as in Definition 1, the* alphabet *of objects and the set of* labels*, respectively; IO is a multiset of objects from A, called* environment*; μ defines the membrane structure, which is a graph, (V, E), where V are vertices, $V \subseteq L$ (the nodes are labels of these compartments), and E edges; $C_1, \ldots, C_n$ are the n compartments of the system – the inner part of each compartment is called the* region *which is delimited by a* membrane*; the labels of the compartments are from L and initial multisets are over A.*

We first discuss various types of rules. It is assumed that the rules below belong to the same compartment, C_i, labeled l_i. Each rule might have a regular expression associated with. When a rule involves more than a compartment, then each compartment might have its own regular expression attached to it. $RE(A \cup \bar{A})$ denotes the set of regular expressions over $A \cup \bar{A}$; each such regular expression defines conditions involving promoters, elements from A, and/or inhibitors, elements from $\bar{A}$. The interpretation of a regular expression $g \in RE(A \cup \bar{A})$, associated with a rule, is that all the promoters appearing in g must be present in the current multiset and none of the inhibitors must appear there. We call this regular expression, g, *guard.* A rule with such a guard is applicable when this is evaluated to true.

A rule can have one of the following types:

- **rewriting and communication** rule: $x \to y \; \{g\}$, where $x \in A^+$, $y \in A^*$, $g \in RE(A \cup \bar{A})$; the right hand side, y, has the form $y = (a_1, t_1) \ldots (a_h, t_h)$, where $a_j \in A$ and $t_j \in L$, $1 \leq j \leq h$, is an object and a target (i.e., the label of a compartment), respectively; the target t_j must be either the label of the current compartment, l_i (more often ignored) or of an existing neighbor of it ($(l_i, t_j) \in E$) or an unspecified one, $*$; otherwise, the rule is not applicable; if a target t_j refers to a label that appears more than once, then one of the involved compartments will be nondeterministically chosen; if t_j is $*$, then the object a_j is sent to a compartment arbitrarily chosen;
- **input-output** rule, is a form of symport/antiport rule: $(x/y) \; \{g\}$, where $x, y \in A^*$, $g \in RE(A \cup \bar{A})$; x from the current region, l_i, is sent to the environment and y from the environment is brought into the current region;

- **system structure rules**; the following types are considered:
  - **membrane division** rule: $[]_{l_i} \rightarrow []_{l_{i_1}} \dots []_{l_{i_h}}$ $\{g\}$, where $g \in RE(A \cup \bar{A})$; the compartment l_i will be replaced by h compartments obtained from l_i, i.e., the content of them will coincide with that of l_i; their labels are $l_{i_1}, \dots, l_{i_h}$, respectively; all the links of l_i are inherited by each of the newly created compartments;
  - **membrane dissolution** rule: $[]_{l_i} \rightarrow \lambda$ $\{g\}$; the compartment l_i will be destroyed together with its links;
  - **link creation** rule: $[]_{l_i}; []_{l_j} \rightarrow []_{l_i} - []_{l_j}$ $\{cg\}$; the current compartment l_i is linked to l_j and, if more than one l_j exists, then one of them will be nondeterministically picked up; cg, called compound guard, describes an expression $l_i.g_1$ *op* $l_j.g_2$, where g_1, g_2 are regular expressions referring to compartments l_i and l_j, respectively; *op* is either *and* or *or*, standing for either both guards are true or at least one is true. If one of the guards is empty then *op* is no longer used; a compound guard defines a Boolean condition across the two compartments;
  - **link destruction** rule: $[]_{l_i} - []_{l_j} \rightarrow []_{l_i}; []_{l_j}$ $\{cg\}$; this is the opposite of link creation and means that compartments l_i, l_j are disconnected; as usual, when more than a link, $(l_i, l_j) \in E$, exists, then only one is considered by this rule; cg is a compound guard.

  The usual behavior of P systems requiring that rewriting and communication, and symport/antiport (input-output) rules are applied in a maximal parallel way (or sequentially in some cases), whereas membrane division, creation, dissolution rules and creation and destruction of links are executed one per membrane, will be considered in this context as well, but in a rather more general way.

The main challenges of this approach are

1. a rigorous definition of the syntax and semantics of kP systems;
2. comparisons between (fragments) of kP systems and well-known variants of P systems;
3. establishing general algorithms to translate different classes of P systems into kP systems (similar to [1, 4]);
4. defining operational semantics for kP systems and providing implementations in model checkers, like Spin, Maude, similar to [3].

Further steps in developing this unified framework might consist of adding other useful modeling features like the possibility of defining rules and compartments using indexes, a certain concept of a module, various other semantics. It is intended to keep the kernel system as generic as possible such that some of the above mentioned extensions will be introduced in a rather syntactical manner allowing to map them into the basic variant, without additional semantics.

Acknowledgement. This work was partially supported by project MuVet, Romanian National Authority for Scientific Research (CNCS, UEFISCDI) grant number PN-II-ID-PCE-2011-3-0688.

References

1. A. Alhazov, L. Pan, Gh. Păun: Trading polarizations for labels in P systems with active membranes. *Acta Informatica*, 41 (2004), 111–144.
2. A. Alhazov, D. Sburlan: Static sorting P systems. In [6], 2006, 215–252.
3. O. Andrei, G. Ciobanu, D. Lucanu, A rewriting logic framework for operational semantics for membrane systems. *Theoretical Computer Sci.*, 373 (2007), 163–181.
4. R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, S. Tini: Membrane systems working in generating and accepting modes: Expressiveness and encodings. *Membrane Computing, 11th International Conference, CMC2010, Jena, Germany, August 2010* (M. Gheorghe et al., eds), LNCS 6501, Springer, 2011, 103–118.
5. R. Ceterchi, C. Martín-Vide: P systems with communication for static sorting. *Pre-Proc. Brainstorming Week on Membrane Computing, Tarragona, February 2003* (M. Cavaliere et al., eds.), Technical Report no 26, Rovira i Virgili Univ., Tarragona, TR 26/03, URV, 2003, 101–117.
6. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*, Springer, 2006.
7. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, 404 (2008), 76–87.
8. R. Nicolescu, M.J. Dinneen, Y.-B. Kim: Structured modelling with hyperdag P systems. Part A. *Membrane Computing, Seventh Brainstorming Week, BWMC 2009, Sevilla, Spain, February 2009* (A.R. Gutiérrez-Escudero at al., eds.), Universidad de Sevilla, 2009, 85–107.
9. The P-lingua Website: `http://www.p-lingua/wiki/index.php/Main_Page`.

18 Bridging P and R

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
Bucharest, Romania, and

Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
`gpaun@us.es`

Some possibilities of bridging MC (P systems) and reaction systems are discussed, the basic idea being of importing ideas from a research area to another one.

Required Notions: cell P system, multiset, reaction system, halting, fypercomputing

Reaction systems (we call them *R systems*) form a recently introduced research area aiming to model the evolution of (bio)chemicals by means of (bio)reactions, in a framework based on the following two fundamental assumptions (we recall them in the formulation from [1]):

The way that we define the result of a set of reactions on a set of elements formalizes the following two assumptions that we made about the chemistry of a cell:

(i) We assume that we have the "threshold" supply of elements (molecules) – either an element is present and then we have "enough" of it, or an element is not present. Therefore we deal with a qualitative rather than quantitative (e.g., multisets) calculus.

(ii) We do not have the "permanence" feature in our model: if nothing happens to an element, then it remains/survives (status quo approach). On the contrary, in our model, an element remains/survives only if there is a reaction sustaining it.

With these postulates in mind, let us consider first some possibilities of passing from R systems to P systems.

Moving from multisets, which are basic in P systems, to sets (actually, to multisets with an infinite multiplicity of their elements, called ω multisets in Section 13) is a fundamental assumption, which changes completely the approach; for instance, we can no longer define computations with the result expressed in terms of counting molecules: the total set of molecules is finite, any molecule is either absent or present in infinitely many copies.

However, as we have mentioned in Section 13, considering P systems with ω multisets leads in an easy way to fypercomputations.

The second assumption of the reaction systems theory (no permanence of objects) looks easier to handle in terms of MC. The immediate idea is to simply remove (by a "deletion rule") any element which does not evolve by means of a reaction; somewhat equivalently, if we want to preserve an object a which is not evolving, we may provide a dummy rule for it, of the type $a \to a$, changing nothing.

Still, many technical problems appear in this framework. The presence of such dummy rules makes the computation endless, while halting is the "standard" way to define successful computations in MC. Moreover, the rules are nondeterministically chosen, hence the dummy rules can interfere with the "computing rules".

While the second difficulty is a purely technical one, the first one can be overpassed by considering other ways of defining the result of a computation in a P system, and there are many suggestions in the literature. We mention here three possibilities: (i) the *local halting* (the computation stops when at least one membrane in the system cannot use any rule), (ii) *signal-objects* (the result consists of the number of objects in a specified membrane at the moment when a distinguished object appears in the system), (iii) *signal-events* (the result consists of the number of objects in a specified membrane at the moment when a distinguished rule is used in the system). Such possibilities were considered in various papers in MC.

Part of these possibilities are checked in [7] from where we recall the following result:

Theorem 2. *Transition P systems of degree 2, using cooperative rules, without the permanence of objects, are computationally complete when the successful computations are defined by local halting or signal-objects. The same result holds true for symport/antiport P systems (of degree 2 and of weight 2) for the case of local halting.*

An interesting *open problem* in this framework is the case of catalytic P systems, known to be universal in the "permanence" assumption.

The case of defining the result of a computation of symport/antiport P systems by means of signals – objects or events – also remains as an *open problem.* (Considering a priority relation on each set of rules can easily solve this problem.) The symport/antiport P system used in the proof of Theorem 2 [7] contains antiport rules of sizes (2, 1) and (1, 2), which is "large" for universality results in the case when objects are persistent. Can the size of rules be decreased also in the case discussed here?

The R systems area has a series of notions of the dynamical systems type which were not too much investigated for P systems (time, events, modules, structure, causality, and so on), and this is also a promising direction of research.

Let us now briefly explore the other direction, from P to R.

The R systems are not meant to define computations, their behavior is deterministic, from a set of symbols we precisely pass to a unique set of symbols. However, starting from an R system, a "generative device" can be defined, based on passing from a configuration to another one (without input from the environment), provided that some nondeterminism is introduced in the R system functioning. Three possibilities of this kind were proposed in [7]: (i) working with tabled R systems, as in Lindenmayer systems (in each step, a table is used, nondeterministically chosen), (ii) considering also a finite multiplicity for some of the objects, and (iii) by introducing a general threshold k on the number of rules which can use the same molecule. All these three possibilities remain to be investigated: properties of the obtained computation graphs, possible links with computing devices from formal language and automata theory, influence of the introduced parameters (number of tables, threshold k), possible hierarchies.

Of course, a general research topic is to find other ways of building a (string or graph) computing device in terms of R systems. A possible question is also the possibility to introduce membranes in the R systems area or other MC ingredients – thus getting a sort of PR systems. (An attempt of this kind is reported in [5], where so-called reaction automata are introduced, but these devices violates both postulates of R systems and use so many ingredients of P systems – multisets, parallelism, nondeterminism, halting – that they are just P automata with a new name.)

References

1. A. Ehrenfeucht, G. Rozenberg: Basic notions of reaction systems, *Proc. DLT 2004* (C.S. Calude et al., eds.), LNCS 3340, Springer, 2004, 27–29.
2. A. Ehrenfeucht, G. Rozenberg: Reaction systems. *Fundamenta Informaticae*, 75 (2007), 263–280.
3. A. Ehrenfeucht, G. Rozenberg: Events and modules in reaction systems. *Theoretical Computer Sci.*, 376 (2007), 3–16.
4. A. Ehrenfeucht, G. Rozenberg: Introducing time in reaction systems. *Theoretical Computer Sci.*, 410 (2009), 310–322.
5. F. Okubo, S. Kobayashi, T. Yokomori: On the properties of languages classes defined by bounded reaction automata. *Theoretical Computer Sci.*, in press.
6. Gh. Păun: Towards fypercomputations (in membrane computing). LNCS, Springer, to appear.
7. Gh. Păun, M.J. Pérez-Jiménez: Towards bridging two cell-inspired models: P systems and R systems. *Theoretical Computer Sci.*, to appear.

19 P Systems and Evolutionary Computing Interactions

Gexiang Zhang

School of Electrical Engineering
Southwest Jiaotong University, Chengdu, P.R. China
zhdxdylan@126.com

Problems related to the so-called membrane algorithms (actually, distributed evolutionary computing, with the distribution controlled by means of membranes, as well as with other MC ingredients used) are mentioned, both in the direction of improving the optimization techniques and in looking for more complex/practical applications.

Required Notions: evolutionary computing, cell P system, active membranes, membrane algorithm

As a relatively young branch of natural computing, MC has gone through thirteen years of intensive research involving areas of theoretical computer science as well as applications in various fields, including systems biology, graphics, linguistics, parallel and distributed computing. However, these applications, in terms of varieties and types, are relatively small compared to a very broad range of applications of evolutionary computing. A natural question would be, whether some combinations of these two models might benefit from the large scope of applications evolutionary computing has already shown so far, and the rigorous and sound theoretical development membrane systems have proved for all its variants.

The possible interplay between MC and evolutionary computation may produce three kinds of research topics:

Membrane-inspired evolutionary algorithms (MIEAs): Since membrane computing was initiated in 1998, a large number of theoretical results, such as various variants of membrane systems and their computational power and efficiency came forth [1]. On the one hand, the way MC is extended into real-world applications is not easy to be addressed and represents an ongoing issue. On the other hand, the hybridization of different computing techniques is an attractive research topic in the area of evolutionary computing, due to a better performance than their counterpart approaches. What can the young paradigm of MC bring to evolutionary computation? Fortunately, MIEAs, formerly called membrane algorithms [2, 3], create a bridge between MC and various real-world applications. MIEA concentrates on generating new evolutionary algorithms for solving optimization problems by using the hierarchical or network structures of membranes and rules of P systems, and the concepts and principles of meta-heuristic search methodologies [3, 4]. The comparative analysis of dynamic behaviors of an instance of MIEAs shows the appropriate combination of MC and evolutionary computation can produce a better capability to balance exploration and exploitation [5], which are two contradictory factors directly related to the performance of an optimization algorithm. Until now, MIEAs have been studied in conjunction with cell P systems with a fixed membrane structure and by considering an evolutionary computing approach as a subalgorithm put inside a membrane [1, 6, 7]. Further research topics are listed below.

1. Consider further combinations of features that make full use of the characteristics of both MC models and evolutionary computing, such as the consideration of cell P systems with active membranes, tissue P systems and population P systems.
2. Usually, in an MIEA an evolutionary algorithm is used as a subalgorithm placed inside a membrane. This idea can be extended. A membrane structure can be used as a framework of the organization of several different types of evolutionary operators, as shown in [8], or several distinct kinds of evolutionary mechanisms, such as a genetic algorithm, evolutionary programming, evolution strategy, differential evolution and particle swarm optimization. Furthermore, the flexible communication rules can be used at the level of genes, instead of at the level of individuals shown in [6, 4].
3. The single-objective problems are usually involved in the investigations reported in the literature. The framework of P systems can offer better population diversity in MIEAs, hence further work can turn to solve problems in a complex environment, such as multi-objective, dynamic, peaked optimization problems, and with/without constraints, to check whether P systems can bring a better performance to evolutionary algorithms.
4. More real-world application problems, such as power system optimization, software/hardware co-design and vehicle route plan, can be solved by using MIEAs.

5. A deep performance analysis and evaluation of MIEAs is necessary to reveal the roles of P systems played in the hybrid optimization algorithms, on the basis of the previous work [5].

Automated design of membrane computing models (ADMCMs): The automated synthesis of some types of MC models or of a high level specification of them is envisaged to be obtained by applying various evolutionary algorithms. ADMCMs aim to circumvent the programmability issue of membrane based models for complex systems [9]. This is quite a complex problem as it involves a great number of parameters (rules, objects, combination of rules) and many semantics associated with P systems.

Membrane evolutionary algorithms (MEAs): MEAs will focus on implementing evolutionary algorithms within a P system environment in order to take advantage of the parallelism and distribution of MC, given that recent investigations are studying the implementation of P systems on parallel or multi-core hardware platforms. An important challenge for any of the above research developments will be to apply them to complex real life systems.

Acknowledgement. This work was supported by the National Natural Science Foundation of China (61170016), the Program for New Century Excellent Talents in University and the Project-sponsored by SRF for ROCS, SEM.

References

1. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
2. T.Y. Nishida: Membrane algorithm with brownian subalgorithm and genetic subalgorithm. *International Journal of Foundations of Computer Science*, 18 (2007), 1353–1360.
3. G.X. Zhang, C.X. Liu, H.N. Rong: Analyzing radar emitter signals with membrane algorithms. *Mathematical and Computer Modelling*, 52 (2010), 1997–2010.
4. G.X. Zhang, J.X. Cheng, M. Gheorghe: A membrane-inspired approximate algorithm for traveling salesman problems. *Romanian Journal of Information Science and Technology*, 14 (2011), 3–19.
5. G.X. Zhang, C.X. Liu, M. Gheorghe: Diversity and convergence analysis of membrane algorithms. *Proc. of the Fifth International Conference on Bio-Inspired Computing: Theories and Application*, 2010, 596–603.
6. G.X. Zhang, M. Gheorghe, C.Z. Wu: A quantum-inspired evolutionary algorithm based on P systems for Knapsack Problem. *Fundamenta Informaticae*, 87 (2008), 93–116.
7. J.X. Cheng, G.X. Zhang, X.X. Zeng: A novel membrane algorithm based on differential evolution for numerical optimization. *International Journal of Unconventional Computing*, 7 (2011), 159–183.
8. G.X. Zhang, M. Gheorghe, Y.Q. Li: A membrane algorithm with quantum-inspired subalgorithms and its application to image processing. *Natural Computing*, 2012, DOI: 10.1007/s11047-012-9320-2. (published online)

9. X.L. Huang, G.X Zhang, H.N. Rong, F. Ipate: Evolutionary design of a simple membrane system. *Proc. CMC 2011, Fontainebleau, France, August 2011* (M. Gheorghe et al., eds.), LNCS 7184, Springer, 2012, 203–214.

20 Metabolic P Systems

Vincenzo Manca
University of Verona, Department of Computer Science, Verona, Italy
vincenzo.manca@univr.it

A dynamical inverse problem for MP systems is formulated, then a possible topological extension of P systems is suggested.

Required Notions: Multiset, multiset rewriting systems, P systems, discrete dynamical systems, metabolism, function approximation.

20.1 A (precise) dynamical problem

A membrane system is a form of compartmentalized rewriting structure based on two main ingredients: multisets of objects and membranes, where multisets of objects and rules are internally placed. Rules transform and move objects among membranes. Metabolic P systems, shortly MP systems, were introduced for modeling real biochemical systems in terms of multiset rewriting. In the last years they have been widely investigated by Verona MNC (Models of Natural Computing and Bioinformatics) group. A brief introduction on MP systems and references can be found in the Scholarpedia page “Metabolic P Systems”.

One of the most recent results about MP systems was the discovery of a methodology for solving dynamical inverse problems, in the sense we are going to explain [2, 3, 4].

A time series $X^T = (X[i] \mid i \leq T \in \mathbb{N})$ is a sequence of real values intended as “equally spaced” in time ($\mathbb{N}$ is the set of natural numbers).

An MP grammar G is a “generator” of time series, determined by the structure ($n, m \in \mathbb{N}$)

$$G = (M, R, I, \Phi),$$

where:

1. $M = \{X_1, X_2, \ldots, X_n\}$ is a finite set of elements called *metabolites.* A *metabolic state* is given by a list of n values, each of which is associated with a metabolite (*parameters* can possibly be added to determine a metabolic state).
2. $R = \{\alpha_j \to \beta_j \mid j = 1, \ldots, m\}$ is a set of *rules*, or *reactions*, with α_j and β_j multisets over M, for $j = 1, \ldots, m$;

3. I are initial values of metabolites, that is, a list $X_1[0], X_2[0], \ldots, X_n[0]$ providing the *metabolic state at step 0*;
4. $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ is a list of functions, called *regulators*, one for each rule, which, for each metabolic state, provide the *fluxes*, the matter quantities consumed/produced by the rules in that state.

An MP graph is a natural graphical representation of G. An MP grammar becomes an MP system when values for the *time interval*, the *population unit*, and the *metabolite masses* are added. An MP grammar G generates the (infinite) time series $(X[i] \mid i \in \mathbb{N})$, from the initial values I, for $X \in \{X_1, X_2, \ldots, X_n\}$, according to the following *Equational Metabolic Algorithm* (EMA), where $\gamma(X)$ denotes the multiplicity of X in the multiset γ and $s[i]$ is the metabolic state at step i:

$$X[i+1] - X[i] = \sum_{j=1}^{m} (\beta_j(X) - \alpha_j(X))\varphi_j(s[i]).$$

DIP formulation for MP: Given n time series $Y_1^T, Y_2^T, \ldots, Y_n^T$, corresponding to some "observed" variables, related by transformation/influence relations among them, find an MP grammar G, expressing the known relations among variables, and generating, for $i \leq T$, exactly, or even approximately enough, the time series $Y_1^T, Y_2^T, \ldots, Y_n^T$.

Many DIP problems of interest for biological/pathological phenomena were solved in terms of MP systems. The solutions obtained resulted from suitable combinations of several ingredients: i) a linear algebra formulation of EMA (to which a *stoichiometric expansion* can be applied in terms of matrix tensor product), ii) the Least Square Evaluation method, iii) the Stepwise Linear Regression method, and iv) some suitable statistical tests based on Fischer's distributions.

A possible field of investigation could concern other classes of DIP problems, in such a way that other kinds of DIP solutions could be found, for these problems, by suitable discrete dynamical systems based on membranes.

20.2 A (vague) topological problem

Membrane computing is based on the intuition of a membrane as a spatial entity closing a subspace (inside/frontier/outside). Cells are the most evident biological realization of this notion. However, if we want to keep this intuition close to the biological reality, the only inclusion relation of membrane containment is too weak. In fact, in the MC literature, some extensions of the original notion of membrane were proposed in terms of tissue-like and neuron-like membrane structures. Even these enrichments are not expressive enough for dealing with aspects were membranes are not framed in a fixed membrane structure hosting computations, but they are subjected to topological transformations exploring and determining forms. This perspective requires a calculus on membranes rather than calculi within, or among, membranes. Some ideas along this line of investigation arose in a research [1] devoted to *multimembranes*, for translating, in a pure membrane setting, computations which can be easily expressed by MP grammars.

A possible field of investigation could concern the formulation of topological (in wide sense) operations among membranes on which calculi can be defined which resemble what happens at the level of morphogenesis in biological systems.

References

1. V. Manca, R. Lombardo: Computing with multi-membranes. *Proc. CMC 2011, Fontainebleau, France, August 2011*, 347–364.
2. V. Manca, L. Marchetti: Metabolic approximation of real periodical functions. *J. Logic and Algebraic Programming*, 79 (2010), 363–373.
3. V. Manca, L. Marchetti: Log-gain stoichiometric stepwise regression for MP systems. *Int. J. Foundations of Computer Science*, 22 (2011), 97–106.
4. V. Manca, L. Marchetti: Solving dynamical inverse problems by means of metabolic P systems. *BioSystems*, 2012, doi:10.1016/j.biosystems.2011.12.006.
5. V. Manca: Metabolic P systems. *Scholarpedia*, 2011.

21 Unraveling Oscillating Structures by Means of P Systems

Thomas Hinze[1,2]

[1]Friedrich Schiller University
Department of Bioinformatics at School of Biology and Pharmacy
Jena, Germany
thomas.hinze@uni-jena.de

[2]Saxon University of Cooperative Education, Dresden, Germany
thomas.hinze@bsa-dresden.de

Issues arising from the use of P systems in modeling biological processes are discussed – especially concerning various circular evolutions of biological processes.

Required Notions: circadian rhythm, cellular dynamics, backtracking

21.1 Motivation

Endogenous oscillations have been identified to be essential for the function of numerous systems found in biology as well as engineering [1]. A common property of these systems lies in their necessity to synchronize and coordinate inherent chemical or physical activities based on periodically iterated trigger signals [4].

Exploration of chronobiological systems emerges as a growing research field within bioinformatics focusing on various applications in medicine, agriculture, and material sciences [8]. Particularly, circadian rhythms embody an interesting biological phenomenon that can be seen as a widespread property of life. The coordination of biological activities into daily cycles provides an important advantage for the fitness of diverse organisms [6]. Based on self-sustained biochemical oscillations, circadian clocks are characterized by a natural period close to but not exactly of 24h that persists under constant conditions (like constant darkness or constant light). Their ability for compensation of temperature variations in the physiological range enables them to maintain the period in case of environmental changes. Furthermore, circadian clocks can be entrained. This property allows a gradual reset of the underlying oscillatory system for adjustment by exposure to external stimuli like daily variations of brightness or daytime-nighttime temperature cycles.

There are numerous types of controllable core oscillators found in circadian clocks. The majority reveals the Goodwin type, a cyclic gene regulatory network composed of mutual activating and inhibiting gene expressions [8]. The most effective way to influence its frequency is modification of protein degradation rates. Furthermore, core oscillators can be of post-translational type [7], exploiting a cyclic scheme of protein phosphorylation, complex formation, or decomposition. Here, the involved catalysts affect the frequency. The third and most complex type of core oscillators includes *compartmental dynamics* [4] aimed to be advantageously modeled using P systems combining a representation of dynamical structures with tracing their spatiotemporal behavior.

21.2 Resulting Challenges

Within the domain of strictly continuous signals quantified by real numbers, modeling and analysis of oscillating behavior has been well-studied [1]. Chemical reaction networks assumed to reside in a homogeneous environment give a typical example: Each species is represented by its concentration which is allowed to vary continuously over time. From the static network topology together with the stoichiometry of the reactions, a corresponding ordinary differential equation system (ODE) can be derived that specifies the reaction rates for each species. Inclusion of parameterized kinetic laws accomplishes a mapping between species concentration and effective reaction rate. The resulting ODE can easily be tested for its capability of undamped oscillating species concentrations. To this end, the *eigenvalues* of the *Jacobian matrix* obtained from the ODE right hand side are sufficient. *Limit cycles* indicate the oscillatory behavior in detail. In case of sinusoidal or almost sinusoidal oscillatory waveforms, even properties of the entrainment behavior can be obtained analytically.

The main advantage of analytical ODE-based methods unequivocally exploits the fact that essential characteristics and properties of a system under study can be directly derived from the underlying mathematical model without any need for a numerical simulation of its dynamical behavior. This makes the evaluation and automated testing of candidate systems resulting from experimental data rather efficient. In contrast, there is currently a lack of corresponding methods within the field of P systems modeled in a discrete manner. Here, system properties mainly emerge from exhaustive simulation studies. Conduction of those studies still requires an extensive amount of human resources. Particularly in case of involved active membranes, compartmental plasticity, and dynamical structures, a toolbox for automated analysis would be helpful. In an ongoing project, we intend to generate sustained oscillatory systems by artificial evolution *in silico* [5]. In this context, the fitness evaluation should answer the question whether the system candidates are able to oscillate endogenously or not and how the periodicity can be controlled. Ideally, this task should be done by a piece of software [3]. Questions concerning a toolbox for systems analysis also coincide with the need to identify appropriate evolutionary operators affecting compartmental structures on the fly. Those operators can be inspired by biological processes found in living cells like division, degeneration, dissolution, creation, separation, merge, endocytosis, exocytosis, or gemmation. Some of these operators can be found in recent frameworks of P systems, but others still lack a detailed specification of their effects to sets of molecular objects and local reactions and transportation rules (configuration update schemes).

21.3 First Ideas

There are different oscillatory scenarios in biological systems. On the one hand, periodicity might also be reflected in temporal changes of the compartmental structure. On the other hand, signalling molecules are often available in low concentrations ranging from single molecules to several thousand copies. Both aforementioned scenarios have in common to prevent pure ODE-based modeling techniques due to the discrete manner of involved key entities. Motivated by the need for an appropriate toolbox covering description, simulation, and analysis of discontinuously considered biological reaction processes, we plan to extend the concept of spatiotemporal P systems with kinetic laws [2, 5] towards an underlying *backtracking mechanism* able to explore the nature of undamped oscillations beyond variations of species concentration. Following the idea of backtracking, the trace of configurations passed by a P system becomes recorded in a suitable way. By monitoring the overall configurations over time, a derivation tree is obtained that provides a comprehensive data pool for further analysis by automated backtracking. Sustained oscillations are expected to appear as recurring, but nonadjacent overall configurations along a path through the derivation tree. In particular, we wish to employ this technique for identification and description of biochemically inspired computational devices equipped with clocks, counters, or frequency scalers. Moreover, we aim for gaining insight into the function of dedicated circadian clocks by reverse engineering using backtracking P systems. This approach could benefit from the flexibility regarding dynamical structures.

References

1. J. Aschoff: A survey on biological rhythms. *Biological Rhythms*, 4 (1981), 3–10.
2. F. Fontana, V. Manca: Discrete solutions to differential equations by metabolic P systems. *Theoretical Computer Science*, 372 (2007), 165–182.
3. G. Gruenert, B. Ibrahim, T. Lenser, M. Lohel, T. Hinze, P. Dittrich: Rule-based spatial modeling with diffusing, geometrically constrained molecules. *BMC Bioinformatics*, 11 (2010), 307.
4. T. Hinze, R. Fassler, T. Lenser, P. Dittrich: Register machine computations on binary numbers by oscillating and catalytic chemical reactions modelled using mass-action kinetics. *International Journal of Foundations of Computer Science*, 20 (2009), 411–426.
5. T. Hinze, R. Fassler, T. Lenser, N. Matsumaru, P. Dittrich: Event-driven metamorphoses of P systems. *Proc. WMC 2009* (D. Corne et al., eds.), LNCS 5391, Springer, 2009, 231–245.
6. T. Hinze, C. Bodenstein, B. Schau, I. Heiland, S. Schuster: Chemical analog computers for clock frequency control based on P modules. *Proc. CMC 2011, Fontainebleau, France, August 2011* (M. Gheorghe et al., eds.), LNCS 7184, Springer, 2012, 182–202.
7. T. Mori, D.R. Williams, M.O. Byrne, X. Qin, M. Egli, H.S. Mchaourab, P.L. Stewart, C.H. Johnson: Elucidating the ticking of an in vitro circadian clockwork. *PLoS Biology*, 5 (2007), 841–853.
8. V.K. Sharma, A. Joshi: Clocks, genes, and evolution. The evolution of circadian organization. *Biological Rhythms* (V. Kumar, ed.), Springer, 2002, 5–23.

22 Simulating Cells Using P Systems

Andrei Păun

Department of Computer Science, Louisiana Tech University, Ruston, USA

Bioinformatics Department, National Institute of Research and Development for Biological Sciences, Bucharest, Romania
apaun@latech.edu, apaun@fmi.unibuc.ro

Possibilities and difficulties encountered in (Ruston group) research on modeling biological processes by means of P systems are discussed, with emphasis on complexity, noise, implementations.

Required Notions: P system models, Gillespie's algorithm, mass action law

To achieve a greater understanding of the biological processes the technology will need to improve and evolve from the current state of "big populations" to discrete events. By big populations we mean the following fact: to be able to perform a specific experiment, the researcher needs a large number of "elements" (say cells) in the same state that is investigated and only then the experiment can reach a conclusion. Once the number of "elements" in the experiment is decreasing, most of our methods to investigate properties of those "elements" become hard/impossible to describe/investigate. Obviously this statement is rather broad and there are some techniques such as FRET analysis that look at discrete events/elements, but we claim that the majority of the current bio-molecular techniques do require large multiplicities of the "element" investigated.

The aforementioned fact has to be understood by the researcher looking to model/simulate cells. It describes the state of the research tools in that area. The modeler can help that particular area by offering better insight into the sub-cellular processes through simulation and prediction. One could immediately point out that since we have a "technological" problem (as stated before) which is precluding us to gain insight into the "discrete" processes, then how can one hope to simulate the sub-cellular mechanisms. The answer is two-fold: (1) cells prove to respond mostly in the same fashion to similar stimuli, meaning that the inherent stochasticity of these systems does not "break" the response pathways (making the simulation from this perspective "easy" as we need to simulate the "important" events, not all the noise associated with the gene regulation mechanisms and their stochasticity); (2) even if we do not know a mechanism, once a model is built based on our best knowledge and we see it diverging from reality in a specific point, we know where to start investigating for other processes/reactions.

There is also a philosophical motivation to using P systems for a cell simulator: P systems were defined to capture the compartmentalized structure of the eukaryotic cells, and indeed this compartmentalization could prove one of the best features of a cell simulator. Furthermore: due to the current biomolecular techniques involving large multiplicities for a species the simulation techniques in the area focussed on ordinary differential equations (ODE) as continuous mathematics both has powerful tools and are easily implemented. But we claim that a continuous mathematics approach in this area of sub-cellular simulation may not be the best approach as some processes have been seen to behave discretely, and in several pathways we can see the multiplicity of some multiprotein complexes appear in very small numbers (below 10). In such cases a discrete simulation technique such as Gillespie's algorithm would be preferable to the simulators based on ODE [4].

Incidentally we have also defined a discrete simulation technique in [2] which was repeatedly improved (see references in [5]) and was lately named NWA with memory. The motivation behind the NWA algorithm was simple: we wanted a discrete mathematics based simulation technique that would be faster than Gillespie's algorithm.

22.1 Brief Description for Current Cellular Models and Simulators

In order to plausibly model the biochemistry of life, individual biochemical interactions need to occur asynchronously over different lengths of time. The model relies on the *law of mass action.* The law states that reaction rate is directly proportional to the number of reactants available in the system. In other words, the time required to execute a rule in the cell is dependent on the number of its reacting species. We note that the rule application is not considered to be instantaneous; the kinetics that are giving the reaction speed model the time required by the molecules involved in the rule to couple together (if the reaction is of second order or higher) as well as the time required for the actual reaction to take place.

The law of mass action gives us the power to temporally describe the evolving configurations of our system. To understand the asynchrony of rule execution, we need to discuss the *kinetic rates* pertaining to the law of mass action. The kinetics of a chemically reactive system are often described as concentration-based values. This is common for the types of experiments used to derive the rates, typically involving enormous populations (millions) of cells. The cells are often lysed as a large population, molecules are measured in terms of light intensity and data are given as concentrations of species across cell population. These values can be averaged across the cell population, yielding concentrations per cell. We rely on these values to fit our models, but the values are derived from entire cell populations instead of individual cells. Hence, the interesting phenotypic, biochemical and physiological characteristics of individual cells can be sometimes overgeneralized (or lost) in lieu of the behavior of the majority of the cells in the population.

Some labs employ techniques to measure single-cell dynamics. For example, interesting results/models on p53 have been reported in [7], where it is shown that individual cells undergo not dampened oscillations, as reported in [1], but each individual cell instead exhibits a different numbers of oscillations. The average behavior for the cell population appeared to be dampened, but individual cells did not behave this way.

We are collaborating with Mark DeCoster's biomedical laboratory from Louisiana Tech University in order to study single cell data via a high-speed imaging system. It is our hope that future collaborations will help unlock some of the secrets behind Fas-induced apoptosis. Regardless of whether data comes from large cell populations or single cell dynamics, we, as modelers, must remain vigilant and build the best models with the data available to us.

Using the law of mass action and discrete kinetic constants we can define the *Waiting Time* (*WT*) of a reaction in the P system. The WT is a value assigned to each reaction, signifying the next timepoint for a single execution of the reaction. As molecular multiplicities will change throughout a simulation, from one configuration to the next, so will the WTs of reactions utilizing those molecules.

We used a min-heap for sorting reactions, where the top of the heap is the reaction with the smallest WT – i.e., the next reaction to be executed. However, we need to use nonstandard methods for maintaining the heap, due to the asynchrony of the rules and the sharing of reactants. These nonstandard methods are similar to those proposed by Gibson and Bruck [3] in their modification to the Gillespie algorithm.

To clarify, when a rule is applied, multiple nodes can have changes to their WT, since the multiplicities of particular species of the system have changed. These species can be shared over multiple reactions. Hence, multiple WT potentially can fail the min-heap property throughout the tree simultaneously at each new configuration. In order to handle this, we use heap maintenance methods similar to those proposed by Gibson and Bruck [3] in their modification of the Gillespie algorithm.

22.2 Improving the Simulators

The following “open problems” are mostly for the simulator developed by our group but should be relevant for other simulators as well.

1. increase the stochasticity at the level of the heap by applying a modified Monte Carlo simulation technique for the first 3 levels of the heap (the fastest 7 reactions),

2. faster implementation such as using C rather than C++ or Java,

3. GPU implementation of the simulator for parallel simulations and identifying “decision points” in the pathway; also running the same model several times could identify the minority from the majority (this information could be lost in an ODE framework for simulation),

4. bigger and better models for sub-cellular mechanisms,

5. using Manca’s Log-gain theory to gather stoichiometric data to be used in simulations [6],

6. implementing the simulation framework as a plug-in in CoPasi for broader dissemination and usage.

Acknowledgements. The author acknowledges support from UEFISCDI – PNII-TE 92/2010.

References

1. R.L. Bar-Or, R. Maya, L.A. Segel, U. Alon, A.J. Levine, M. Oren: Generation of oscillations by the p53-Mdm2 feedback loop: A theoretical and experimental study. *Proc. Natl. Acad. Sci., USA*, 97 (2000), 11250–11255.
2. S. Cheruku, A. Păun, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra: Simulating FAS-induced apoptosis by using P systems. *Progress in Natural Science*, 17 (2007), 424–431.
3. M.A. Gibson, J. Bruck: Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104 (2000), 1876–1889.
4. D.T. Gillespie: Exact stochastic simulation of coupled chemical Reactions,” *The Journal of Physical Chemistry*, vol. 81, no. 25, 1977, pp. 2340–2361.
5. J. Jack, A. Păun: Discrete modeling of biochemical signaling with memory enhancement. *LNBI Transactions on Computational Systems Biology*, 5750 (2009), 200–215.
6. V. Manca: The metabolic algorithm for P systems: Principles and applications. *Theoretical Computer Science*, 404 (2008), 142–155.
7. G. Lahav, N. Rosenfeld, A. Sigal, N. Geva-Zatorsky, A.J. Levine, M.B. Elowitz, U. Alon: Dynamics of the p53-Mdm2 feedback loop in individual cells. *Nature Genetics*, 36 (2004), 147–150.

23 P Systems for Computational Systems and Synthetic Biology

Marian Gheorghe[1,2], **Vincenzo Manca**[3],
Francisco-José Romero-Campero[4]

[1]Department of Computer Science, University of Piteşti, Romania

[2]Department of Computer Science, University of Sheffield, UK
m.gheorghe@dcs.shef.ac.uk

[3]Department of Computer Science, University of Verona, Italy
vincenzo.manca@univr.it

[4]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
fran@us.es

Deterministic and stochastic P system models are discussed in the context of specifying fairly complex biological systems; their usage for systems and synthetic biology is also presented.

Required Notions: metabolic P systems, dynamical inverse problem, stochastic P systems, Gillespie algorithm, systems biology, synthetic biology

The approaches based on P systems aiming to provide coherent descriptions of fairly complex biological systems are either deterministic or stochastic [5]. Two such variants are discussed below, but some more variants of the above mentioned types of P systems are available in the current literature, see [15] and Sections 21 and 22 of this paper.

Metabolic P systems (MP systems for short) were introduced in 2004 as a particular kind of P systems devised for modeling metabolic processes [7]. Their main goal consists in solving dynamical inverse problems (DIPs) by means of discrete systems. A general algorithm, called Log-Gain Stoichiometric Stepwise Regression (LGSS), providing MP solutions to DIPs was obtained, in a systematic way, by integrating finite difference recurrent equations, least square method, stepwise regression, and related Fisher tests, within a suitable linear algebra framework where solutions can be expressed as ordinary and tensor products among matrices [11]. A MATLAB implementation of LGSS was developed by Luca Marchetti [12].

Many successful applications of MP theory to biological dynamics were developed, starting from classical examples (Lotka-Volterra, Brusselator, Mitotic Oscillator) [10]. Presently, the two main applications under investigation concern the insulin-glucose dynamics in diabetes pathologies and genetic expression in a kind of breast cancer (in cooperation with endocrinologists and clinicians in Italy, Verona and in USA, Detroit). A synthetic description and references is given by Vincenzo Manca in [8, 9].

Stochastic P systems, SP system for short, are rule-based discrete and stochastic multicompartmental systems used as abstract structures to model stochastic cellular systems [14]. The key difference between the original P systems and SP systems consists in a stochastic constant that is specifically associated with each rule. This constant is used to determine in a specific state or configuration of the system the probability of applying the corresponding rule and the time elapsed between rule applications according to Gillespie's stochastic simulation algorithm [6].

SP systems allow the incremental and parsimonious design of models by providing modelers with the feature of modularity explicitly [4]. A P system module consists of a finite set of rewriting rules that may contain some free variables in their objects, labels and stochastic constants. Modules can be arranged in libraries so they can be reused to define the rewriting rules of different models. In this respect, modules act like macros that get expanded once the corresponding module variables are instantiated with specific molecular species names, numerical values for the stochastic constants and compartment names.

A variant of SP systems, *lattice population P systems* [18], allow modelers to represent multi-cellular systems with specific geometries by distributing copies of given individual stochastic P systems over the points of a finite geometrical lattice.

SP systems have been implemented in the software tool for the specification, simulation, analysis and optimization of systems and synthetic biology models, Infobiotics workbench [3].

These systems have been used to model signal transduction pathways [13, 1], bacterial gene regulation [17], bacterial populations [16], metapopulations [2] and synthetic biology problems [19].

Membrane computing has made very significant contributions in certain areas of computer science and has produced some impact with respect to a number of applications. It remains a challenge to show how it copes with complex applications, especially in systems and synthetic biology. Some of these challenges are listed below:

- identify more complex systems to be specified by one of the variants of P systems described above or presented in [15];
- extend the current variants with additional features in order to cope with more complex applications;
- create a repository of illustrative biological case studies;
- develop additional complementary approaches that help analyzing biological systems – data sensitivity analysis, property data extraction and verification, hierarchies of languages allowing to map P system specifications into biochemical reactions;
- implement adequate tools exploiting the latest technologies and create benchmark problems to assess them.

Acknowledgement. M.G.'s work was partially supported by project MuVet, Romanian National Authority for Scientific Research (CNCS, UEFISCDI) grant number PN-II-ID-PCE-2011-3-0688.

References

1. D. Besozzi, P Cazzaniga, S. Cocolo, G. Mauri, D. Pescini: Modelling diffusion in a signal transduction pathway: the use of virtual volumes in P systems. *Int. J. Found. Comput. Sci.*, 22 (2011), 89–96.
2. D. Besozzi, P Cazzaniga, D. Pescini, G. Mauri: Modelling metapopulations with stochastic membrane systems. *BioSystems*, 91 (2008), 499–514.
3. J. Blakes, J. Twycross, F.J. Romero-Campero, N. Krasnogor: The Infobiotics Workbench: an integrated in silico modelling platform for Systems and Synthetic Biology. *Bioinformatics*, 27 (2011), 3323–3324.
4. H. Cao, F.J. Romero-Campero, S. Heeb, M. Cámara, N. Krasnogor: Evolving cell models for systems and synthetic biology. *Syst. Synth. Biol.*, 4 (2010), 55–84
5. M. Gheorghe, V. Manca, F.J. Romero-Campero: Deterministic and stochastic P systems for modelling cellular processes. *Natural Computing*, 9 (2010), 457–473.
6. D.T. Gillespie: Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58 (2007), 35–55.
7. V. Manca: The metabolic algorithm for P systems: Principles and applications.*Theoretical Computer Science*, 404 (2008), 142–155.
8. V. Manca: Metabolic P systems.*Scholarpedia*, 6 (2010), 9273.
9. V. Manca: Fundamentals of metabolic P systems. Chapter 6 in [15], 475–498.
10. V. Manca L. Bianco, F. Fontana: Evolutions and oscillations of P systems: Theoretical considerations and application to biological phenomena. *Proc. WMC 2004, Milan, Italy, June 2004*, LNCS 3365, Springer, 2005, 63–84.
11. V. Manca, L. Marchetti: Solving dynamical inverse problems by means of metabolic P systems. *BioSystems*, to appear. DOI:10.1016/j.biosystems.2011.12.006.
12. L. Marchetti, V. Manca: A methodology based on MP theory for gene expression analysis. *Proc. CMC 2011, Fontainebleau, France, August 2011*, LNCS 7184, Springer, 2012, 300–313.
13. A. Păun, M.J. Pérez-Jiménez, F.J. Romero-Campero: Modeling signal transduction using P systems. *Proc. WMC 2006, Leiden, The Netherlands, July 2006*, LNCS 4361, Springer, 2006, 100–122.
14. Gh. Păun, F.J. Romero-Campero: Membrane computing as a modeling framework: cellular systems case studies. *Proc. Formal Methods for Computational Biology, 8th Intern. School*, Bertinoro, Italy, June 2008 (M. Bernardo et al, eds.), LNCS 5012, Springer, 2008, 168–214.
15. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing.* Oxford Univ. Press, 2010.
16. F.J. Romero-Campero, M.J. Pérez-Jiménez: A model of the quorum sensing system in Vibrio fischeri using P systems. *Artificial Life*, 14 (2008), 95–109.
17. F.J. Romero-Campero, M.J. Pérez-Jiménez: Modelling gene expression control using P systems: The Lac Operon, a case study. *BioSystems*, 91 (2008), 438–457.
18. F.J. Romero-Campero, J. Twycross, M. Cámara, M. Bennett, M. Gheorghe, N. Krasnogor: Modular assembly of cell systems biology models using P systems. *Int. J. Found. Comput. Sci.*, 20 (2009), 427–442.

19. J. Smaldon, F.J. Romero-Campero, F. Fernández Trillo, M. Gheorghe, C. Alexander, N. Krasnogor: A computational study of liposome logic: towards cellular computing from the bottom up. *Syst. Synth. Biol.*, 4 (2010), 157–179.

24 Biologically Plausible Applications of SN P Systems for an Explanation of Brain Cognitive Functions

Adam Obtulowicz

Institute of Mathematics, Polish Academy of Sciences, Warsaw, Poland
A.Obtulowicz@impan.pl

Some conjectures about the possibility of using SN P systems and extension of them for modeling features of the brain (such as learning, modularity) are formulated.

Required Notions: spiking neuron, SN P system, learning

The (hierarchical) clustering (scene segmentation in particular) and binding (feature integration) problem solution in cortical neural networks together with cortical subnetworks realizing Radial Basic Functions (briefly RBFs) represent, among others, the cognitive functioning of brain. Recently, various network models of clustering, binding problem solution, and realization of RBFs in cortical networks have been proposed, where spiking neural networks are the most biologically plausible models, see [16], [17], [2], [3], [12], [14], [15], and [11] for a review. The main common feature of these models is Hebbian learning which provides their biological evidence. On the other hand, a transformation of an idea of Hebbian learning from a framework of spiking neural networks to a framework of SN P systems (cf. [10]) has been proposed in [8]. Thus, one formulates the following question:

> Do SN P systems provide biologically plausible mathematical models of brain cognitive functions?

We approach the question and an answer to it by the following discussion of conjectures and setting open problems.

Papers [5], [9] contain promising applications of SN P systems for solving topic problems related to some cognitive brain functions. But biological evidence of these applications seems problematic because Hebbian learning procedures approach is not considered for them.

On the other hand, the Hebbian learning modeled by SN P systems with only input neurons and one output neuron presented in [8] and solution of XOR problem by spiking neural networks equipped with a Hebbian learning procedure and with only three input neurons and one output neuron described in [4] gives rise to the following conjecture:

Conjecture 1. *There exists a learning problem, understood as in* [8]*, whose output is an SN P system solving XOR problem.*

If we compare precise timing of spikes approach for spiking neural networks to the number of spikes approach for SN P systems, then the latter seems coarse and hence less biologically plausible than the spiking neural network approach.

On the other hand, the precise timing of spikes approach for spiking neural networks is less biologically plausible than probabilistic spiking neural networks because a relevant amount of noise is contained in the behavior of neurons (cf. [7]). Therefore it is worth to initiate a research of probabilistic SN P systems.

The view that human mind is “massively modular” (cf. [6], [13]) argued by massively parallel functioning of brain neural network modules, gives rise to a question of approaching these massive modularity and massive parallelism of mind and brain by application of a concept of a network of communicating SN P systems equipped with Hebbian learning procedures, respectively. The SN P systems constituting that network could correspond to brain network modules realizing simultaneously various cognitive functions, respectively.

On the other hand, since SN P systems seem more coarse with respect to an approach to time than spiking neural networks with precise timing of spikes, like, e.g., in [2], we propose the following conjecture.

Conjecture 2. *A biologically plausible modularity of brain could be represented* (*modeled*) *by the following hybrid constructs*:

1. *a two-level construct of a* spiking super-neural P system *which is an SN P system whose neurons are superneurons, i.e., multi-layer spiking neural networks with a precise timing of spikes like, e.g., in* [2]*,*
2. *a three-level construct of a* spiking sub-super-neural P system *which is a spiking super-neural P system as above, where the neurons of superneurons are P systems approaching neurons as cells which produce and transport copies of molecules between electrically charged membranes.*

The construct in 1) gives rise to multi-layer spiking networks which could learn themselves like in [2] their modular structure of spiking super-neural P systems and hence which could explain emergence of cognitive capabilities of brain.

It is worth to discuss the above constructs with regard to the possibility of their molecular implementation which is suggested by recent findings outlined in [1].

References

1. A. Bandyopadhyay, D. Fujita, R. Pati: Architecture of a massively parallel processing nano-brain operating 100 billion molecular neurons simultaneously. *International Journal of Nanotechnology and Molecular Computation*, 1 (2009), 50–80.

2. S.M. Bohte: *Spiking Neural Networks.* Professorschrift, Leiden University, 2003.
3. O. Booij: *Temporal Pattern Classification using Spiking Neural Networks.* M.Sc. Thesis, Amsterdam University 2004.
4. O. Booij, Hieu tat Nguyen: A gradient descent rule for spiking neurons emitting multiple spikes. *Applications of Spiking Neural Networks* (S.M. Bohte, J.N. Kok, eds.), *Information Processing Letters*, Amsterdam, 2005.
5. R. Ceterchi, I.A. Tomescu: Spiking neural P systems–a natural model for sorting networks. *Proc. Sixth Brainstorming Week on Membrane Computing* (D. Diaz-Pernil et al., eds.), Sevilla, February 4–8, 2008, RGNC Report 01/2008, Fenix Editora, Sevilla, 2008, 93–105.
6. D. Geary: *The Origin of Mind: Evolution of Brain, Cognition, and General Intelligence.* American Psychological Association 2005.
7. W. Gerstner: Population dynamics of spiking neurons: Fast transients, asynchronous states, and locking. *Neural Computation*, 12 (2000), 43–89.
8. M.A. Gutiérez-Naranjo, M.J. Pérez-Jiménez: A spiking neural P systems based model for Hebbian learning. *Proc. 9th Workshop on Membrane Computing* (P. Frisco et al., eds.), Edinburgh, July 28–31, 2008, 189–207.
9. M. Ionescu, D. Sburlan: Some applications of spiking neural P systems. *Proc. 8th Workshop on Membrane Computing* (Eleftherakis et al., eds.), Thessaloniki, June 25–28, 2007, 383–394.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fund. Inform.*, 71 (2006), 279–308.
11. A. Kasiński, F. Ponulak: Comparison of supervised learning methods for spike time coding in spiking neural networks. *Int. J. Appl. Math. Comput. Sci.*, 16 (2006), 101–113.
12. A. Knoblauch, G. Palm: Scene segmentation by spike synchronization in reciprocally connected visual areas. II: Global assemblies and synchronization on larger space and time scales. *Biol. Cybern.*, 87 (2002), 168–184.
13. K. MacDonald, D. Chiappe: Review of [6] in *Human Ethology Bulletin*, 21 (2006), 14–18.
14. B. Meftah, A. Benyettou, O. Lezoray, W. Qingxiang: Image clustering with spiking neuron network. *World Congress on Computational Intelligence, International Joint Conference on Neural Networks*, Hong-Kong, 2008.
15. S.C. Moore: *Back-propagation in Spiking Neural Networks.* M.Sc. Thesis, University of Bath 2002, `http://www.simonchristianmoore.co.uk/Thesis4.html`.
16. T. Natschläger, B. Ruf: Spatial and temporal pattern analysis via spiking neurons. *Network: Comp. Neural Systems*, 9 (1998), 319–332.
17. B. Ruf: *Computing and Learning with Spiking Neurons–Theory and Simulation.* Doctoral Thesis, Technische Universität Graz, 1998.

25 Computer Vision

Daniel Díaz-Pernil[1], **Miguel A. Gutiérrez-Naranjo**[2]

[1] CATAM Research Group, Dept. of Applied Mathematics I
University of Sevilla, Spain
sbdani@us.es

[2] Research Group on Natural Computing, Dept. of Computer Science and AI
University of Sevilla, Spain
magutier@us.es

Some possibilities to employ MC techniques in computer vision (especially in thresholding, smoothing, homology theory) are discussed.

Required Notions: array grammar, array-rewriting P system, cell and tissue P systems

Computer vision is probably one of the challenges for computer scientists in the next years. From a biological point of view, vision is an extremely complex process involving the transformation of the light energy into a signal which leaves the eye by way of the optic nerve and arrives to the brain, where it is interpreted. From a computational point of view, a digital image is a function from a two dimensional surface which maps each point form the surface to a set of features as bright or color.

In MC, there is a large tradition in handling information structured as two dimensional objects (see, e.g., [2, 3, 9, 16]). The main motivation for these studies is to bring together P systems and picture grammars. From a technical point of view, arrays are two-dimensional objects placed inside the membranes as strings are one-dimensional objects in the model of P systems with string objects [13].

In [3], the model of array-rewriting P systems was presented on the basis of the transition P systems: Rules are of type $\mathcal{A} \to \mathcal{B}(tar)$ where $\mathcal{A}$ is the array to be rewritten, $\mathcal{B}$ is the new one, and $tar \in \{here, in, out\}$ indicates the place of the picture after the substitution has been made.

Recently, a new research line has been open by applying well-known MC techniques for solving problems from digital imagery. For example, 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. Segmentation has shown its utility, for example, in bordering tumors and other pathologies or computer-guided surgery. In [5, 8, 10, 11] we can find several approaches to this problem with MC techniques. Other problems, as *thresholding* [4] or *smoothing* [18] have also been considered in the framework of MC. Special attention deserves [14], where the *symmetric dynamic programming stereo* (SDPS) algorithm [15] for stereo matching was implemented by using simple P modules with duplex channels.

A different approach to computer vision can also be obtained from computational topology. In particular, algebraic topology provides techniques and algorithms for handling digital images from a topological point of view. Recently, the links between algebraic topology and MC have started to be explored via *homology theory* [6, 7, 12]. *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. 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 a similar way with other applications of P systems, the theoretical advantages of the MC techniques for computer vision need a powerful software and hardware for an effective implementation. The use of these new technologies for the parallel implementation of P systems techniques applied to computer vision have started to be explored with promising experimental results [1, 17, 18].

An appropriate combination of MC techniques together with an efficient parallel implementation on the new hardware architectures can provide competitive algorithms to different problems from computer vision. Among them, we can cite dealing with textures, colors and/or 3D objects (or even 4D objects, where the evolution of objects in *time* is also considered). From algebraic topology, the calculus of complex topological invariants of 2D and 3D objects can be a source of new open problems for MC.

References

1. J. Carnero, D. Díaz-Pernil, M.A. Gutiérrez-Naranjo: Designing tissue-like P systems for image segmentation on parallel architectures. *Ninth Brainstorming Week on Membrane Computing* (M.A. Martínez del Amor et al., eds.), Fénix Editora, Sevilla, 2011, 43–62
2. R. Ceterchi, R. Gramatovici, N. Jonoska, K.G. Subramanian: Tissue-like P systems with active membranes for picture generation. *Fundamenta Informaticae*, 56 (2003), 311–328.
3. R. Ceterchi, M. Mutyam, Gh. Păun, K.G. Subramanian: Array-rewriting P systems. *Natural Computing*, 2 (2003), 229–249.
4. H.A. Christinal, D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Thresholding of 2D images with cell-like P systems. *Romanian Journal of Information Science and Technology*, 13 (2010), 131–140.
5. H.A. Christinal, D. Díaz-Pernil, P. Real: Segmentation in 2D and 3D image using tissue-like P system. LNCS 5856, Springer, 2009, 169–176.
6. H.A. Christinal, D. Díaz-Pernil, P. Real: Using membrane computing for obtaining homology groups of binary 2D digital images. LNCS 5852, Springer, Berlin, 2009, 383–396.
7. H.A. Christinal, D. Díaz-Pernil, P. Real: P systems and computational algebraic topology. *Mathematical and Computer Modelling*, 52 (2010), 1982–1996.
8. H.A. Christinal, D. Díaz-Pernil, P. Real: Region-based segmentation of 2D and 3D images with tissue-like P systems. *Pattern Recognition Letters*, 32 (2011), 2206–2212.
9. K.S. Dersanambika, K. Krithivasan: Contextual array P systems. *Intern. Journal of Computer Mathematics*, 81 (2004), 955–969.

10. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, H. Molina-Abril, P. Real: A bio-inspired software for segmenting digital images. *Proc. Fifth International Conference on Bio-Inspired Computing. Theories and Applications* BIC-TA (A.K. Nagar et al., eds.), vol. 2 (2010), 1377–1381.
11. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, H. Molina-Abril, P. Real: Designing a new software tool for digital imagery based on P systems. *Natural Computing*, 2011 `http://dx.doi.org/10.1007/s11047-011-9287-4`.
12. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, P. Real, V. Sánchez-Canales,: Computing homology groups in binary 2D imagery by tissue-like P systems. *Romanian Journal of Information Science and Technology*, 13 (2010), 141–152.
13. C. Ferretti, G. Mauri, C. Zandron: P systems with string objects. Chapter 7 of *The Oxford Handbook of Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2010, 168–197.
14. G. Gimel'farb, R. Nicolescu, S. Ragavan: P systems in stereo matching. LNCS 6855, Springer, Berlin, 2011, 285–292.
15. G. Gimel'farb: Probabilistic regularisation and symmetry in binocular dynamic programming stereo. *Pattern Recognition Letters*, 23 (2002), 431–442.
16. S.N. Krishna, R. Rama, K. Krithivasan: P systems with picture objects. *Acta Cybernetica*, 15 (2001), 53–74.
17. F. Peña-Cantillana, D. Díaz-Pernil, A. Berciano, M.A. Gutiérrez-Naranjo: A parallel implementation of the thresholding problem by using tissue-like P systems. LNCS 6855, Springer, 2011, 277–284.
18. F. Peña-Cantillana, D. Díaz-Pernil, H.A. Christinal, M.A. Gutiérrez-Naranjo: Implementation on CUDA of the smoothing problem with tissue-like P systems. *Intern. Journal of Natural Computing Research*, 2 (2011), 25–34.

26 Open Problems on Simulation of Membrane Computing Models

Manuel García-Quismondo, Luis F. Macías-Ramos,
Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado,
Luis Valencia-Cabrera

Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence, University of Sevilla, Spain
{mgarciaquismondo,lfmaciasr,mdelamor,perezh,lvalencia}@us.es

Research ideas related to the extension of the P-lingua dedicated languages and on the implementation of P systems on reconfigurable or parallel hardware (e.g., NVIDIA architectures) are mentioned.

Required Notions: P system models, GPU computing, P-Lingua, MeCoSim

The development of P system simulators, and of other related software tools, becomes a critical point in the processes of model validation and virtual experimentation. For this purpose, a software framework for specifying and simulating

P systems, called P-Lingua, was developed [7]. Moreover, a generic software to generate graphical applications based on P-Lingua, called MeCoSim, was also developed. Finally, in order to accelerate the simulation by implementing P systems parallelism on high performance platforms, some simulators were developed by using GPU computing [5]. Research in all these directions are under development.

(A) Simulation Framework: P-Lingua and PLinguaCore

P–Lingua has been successfully applied to ecosystem modeling problems [6], formal model checking and to solve computationally hard problems [7]. It supports several P system models, such as *active membrane* models [7], *Tissue* P system models and Spiking Neural P Systems (*SN P*) [9] systems.

Nevertheless, there is still plenty to do in order to extend the capabilities of P-Lingua. Both expressivity and functionality issues should be improved to renew the P-Lingua menu and attract new users. First, inclusion of parsing directives should be implemented in order to, say, modify the behavior of existing models. A fast learner example would be the 'asynchronous' behavior, that is, cracking the universal clock that reigns the computation process in most of standard models (this is already included for the case of SN P Systems). Then, integration of new models, such as numerical P systems and some specific types of SN P Systems, incorporating *weights* and *astrocytes with their different flavors*, remains unexplored. Finally, re-factoring of the work done to bring some exotic elements of the *reaction systems.*

(B) Generic end-user graphical applications: MeCoSim

In the last few years, there have been some interesting, user-friendly, successful software applications for modeling and simulating P systems, mainly focused on biological systems: *MetaPlab* [4] for internal mechanisms of biological systems by means of MP System; *BioSimWare* [1] and *Infobiotics* [2] for P system based multicompartmental stochastic simulations of complex biological systems, the last one including Synthetic Biology; and *EcoSim* [6], a family of probabilistic simulators for different ecosystems.

However, a general application for the study, analysis, modeling, visual simulation, model checking, optimization of as many as possible variants of P systems has not been provided. A first approach has been developed with *MeCoSim* [10]. Some plugins have been developed to provide some analysis and model checking capabilities. It has been successfully applied as an assistant tool for the iterative design of ecosystem models, and to solve computationally hard problems by using tissue and SN P system models.

Nevertheless, there are many challenges to solve. The core of the visual application should include more analysis and modeling tools to ease the work of the P systems designer. Also, some *interfaces* to communicate with different simulation engines should be developed to run simulation against simulators implemented in different local or remote platforms and architectures. It should integrate with different applications for formal model checking, enabling the user to extract and/or validate properties of the studied models. Eventually, new P systems *models* should be added to MeCoSim, providing the general functionalities and new possible plugins to many potential P systems designers.

(C) Simulation on High Performance Platforms: GPU computing

So far, there have been many efforts on the development of GPGPU based simulators for P systems. In fact, the following P systems models have been successfully simulated by means of GPUs: P systems with active membranes and division rules [5], a family of P systems with active membranes solving `SAT` in linear time [5], SN P systems (SNP) with and without delays [3], and ENPSs [8].

On the other hand, there exist many other models which, to the best of our knowledge, are yet to be simulated by means of GPGPU. These models include tissue P systems, population dynamics P systems, stochastic P systems, hyperdag P systems, numerical P systems and string P systems, to name just a few.

Another challenge is the *integration* of GPU simulators on end-user MC software frameworks. Although some steps have been taken in this direction with the P-Lingua automatic generation [7] of P system files to be parsed by GPU simulators [5], there is still a long way to walk for an efficient interaction between these two kinds of technological tools.

Last but not least, a thorough *performance comparison* between GPU simulators and other HPC approaches is yet to be developed. These approaches include reconfigurable hardware (FGPA, DSP), computer clusters with OpenMP and MPI, etc. The need for some works on this direction has been previously noticed [5, 8]. Finally, it is also interesting to port current developments of GPU simulators to the last GPU platforms, based on both NVIDIA and AMD ATI architectures, and on GPU based clusters.

Acknowledgements. The authors are supported by the project TIN 2009-13192 from “Ministerio de Economía y Competitividad” of Spain, and by “Proyecto de Excelencia con Investigador de Reconocida Valía” P08-TIC-04200 from Junta de Andalucía, both co-financed by FEDER funds. Manuel García-Quismondo is also supported by the National FPU Grant Programme from the Spanish Ministry of Education.

References

1. D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini: BioSimWare: A software for the modeling, simulation and analysis of biological systems. LNCS 6501, Springer, 2011, 119–143.
2. J. Blakes, J. Twycross, F.J. Romero–Campero, N. Krasnogor: The infobiotics workbench: an integrated in silico modelling platform for Systems and Synthetic Biology. *Bioinformatics*, 27 (2011), 3323–3324.
3. F.C. Cabarle, H. Adorna, M.A. Martínez-del-Amor: A spiking neural P system simulator based on CUDA. LNCS 7184, Springer, Berlin, 2012, 87–103.
4. A. Castellini, V. Manca: MetaPlab: A computational framework for metabolic P systems. *Pre-proceedings of WMC'08*, 2008, Edinburgh, Scotland.

5. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez: Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics*, 11 (2010), 313–322.
6. M.A. Colomer, A. Margalida, D. Sanuy, M.J. Pérez–Jiménez: A bio–inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. *Ecological modelling*, 222 (2011), 33–47.
7. M. García-Quismondo, R. Gutiérrez-Escudero, M.A. Martínez-del-Amor, E. Orejuela-Pinedo, I. Pérez-Hurtado: P-Lingua 2.0: A software framework for cell-like P systems. *International Journal of Computers, Communications and Control*, 4 (2009), 234–243.
8. M. García-Quismondo, M.J. Pérez–Jiménez, L.F. Macías–Ramos: Implementing ENPS by means of GPUs for AI applications. *Beyond AI. Interdisciplinary Aspects of Artificial Intelligence (BAI 2011)*, 08/12/2011–09/12/2011, Pilsen, Czech Republic.
9. L.F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M.J. Pérez-Jiménez, A. Riscos-Núñez: A P-Lingua based simulator for spiking neural P systems. LNCS 7184, Springer, 2012, 257–281.
10. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez: MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems. *IEEE Proceedings of BIC-TA 2010*, I, 637–643.

Closing Remarks

As also said in the Introduction, this collection of open problems and research topics in MC was initially meant to be a working material, for 10th BWMC, and it was updated and completed several times. However, no such list can be complete, neither uniform, in what concerns the type of problems, their technicality, difficulty, range of interest. As expected, some problems are local, others are very general, while the sections are not at all uniform in style (we have preserved in a great extent the contributors writing). Moreover, many further research ideas wait to be addressed in MC, for instance, in the P and dP automata area, the SN P systems area, complexity, dynamical systems approach – not to speak about applications (from biology and bio-medicine, to ecology, robot control, approximate optimization). Still, we believe that such a list is useful, on the one hand, because it can entail cooperation about the co-authors of the paper and the readers, and, on the other hand, because it points out active research areas of MC, indicating its "frontiers". Actually, this "mega-paper" proved already to be useful during the 10th BWMC, where several of the proposed research topics were addressed – the paper incorporates some changes due to these recent progresses.

Acknowledgements

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. The work of M. Gheorghe was partially supported by project

MuVet, Romanian National Authority for Scientific Research (CNCS, UEFISCDI) grant number PN-II-ID-PCE-2011-3-0688.

The editors are much indebted to all MC researchers who have contributed to this “mega-paper”, to Grzegorz Rozenberg for many suggestions during the preparation of the paper, as well to two anonymous referees for carefully reading a previous version of the text.

A Formal Framework for Clock-free Networks of Cells

Sergiu Ivanov

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
sivanov@math.md
University of the Academy of Science of Moldova,
Faculty of Real Sciences,
3/2, Academiei str., MD-2028, Chişinău, Republic of Moldova

1 Introduction

Clock-free P systems were introduced in [7] as a natural extension to transitional membrane systems. The idea sparks from the observation of the fundamental difference between how transitional P systems evolve and how processes take place in biological cells: transitional P systems evolve in a series of crisp evolution steps, under the control of a global clock, while their biological prototypes have nothing similar to such a device.

In clock-free P systems, the attempt is made to bridge this difference by discarding any global step synchronisation mechanism. Any rule application lasts differently and there is no way of knowing when exactly the right-hand side of a rule will be added to the system.

The clock-free model produces two intuitive impressions. Firstly, it seems to be much closer to the real-world processes in the cell: the duration of clock-free rules is not regulated by any external mechanism and is expressed as a real number. Secondly, the absence of any built-in global step synchronisation seems to be very specific and quite unwieldy to manage. In fact, one of the best-working approaches to producing meaningful results with clock-free P systems is cutting down on parallelism as much as possible: this is how the computational completeness of these devices is shown in [7].

There are other ways to introduce time into P systems, an example could be timed P systems (see [1]). In this model, however, the global clock is still present.

In this paper we provide a formal description of the semantics of clock-free P systems, or rather the more general concept of clock-free networks of cells, and show how these devices can be modelled in transitional P systems.

This paper is heavily based on [3]. While we tried to introduce all the relevant concepts in this paper as well, getting acquainted with [3] would still be recommended.

2 Preliminaries

2.1 Multisets

Given a finite set A, by $|A|$ we understand the number of elements in A.

Let V be a finite alphabet; then V^* is the set of all finite strings of a V, and $V^+ = V^* - \{\lambda\}$, where λ is the empty string. By $\mathbb{N}$ we denote the set of all non-negative integers, by $\mathbb{N}^k$ – the set of all vectors of non-negative integers.

Let V be a finite set, $V = \{a_1, \ldots, a_k\}$, $k \in \mathbb{N}$. A *finite multiset* M over V is a mapping $M : V \to \mathbb{N}$. For each $a \in V$, $M(a)$ indicates the number of "occurrences" of a in M. The value $M(a)$ is called the *multiplicity* of a in M. The size of the multiset M is $|M| = \sum_{a \in V} M(a)$, i.e., the total count of the entries of the multiset. A multiset M over V can also be represented by any string x which contains exactly $M(a_i)$ instances of a_i, $1 \le i \le k$. The support of M is the set $supp(M) = \{a \in V \mid M(a) \ge 1\}$, which is the set which contains all elements of the multiset. For example, the multiset over $\{a, b, c\}$ defined by the mapping $\{a \to 3, b \to 1, c \to 0\}$ can be written as $a^3 b$. The support of this multiset is $\{a, b\}$.

The class of all finite multisets over V is denoted by $\langle V, \mathbb{N}\rangle$. One may also consider mappings M of the form $M : V \to \mathbb{N}_\infty$, where $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, i.e., the elements may have infinite multiplicity. A multiset M is infinite if $\big(\exists i \in \mathbb{N}\big)\big(1 \le i \le k\big)\big(M(a_i) = \infty\big)$, i.e., at least one element is of infinite multiplicity. The class of multisets M over V with $M : V \to \mathbb{N}_\infty$ is denoted by $\langle V, \mathbb{N}_\infty\rangle$. For $W \subseteq V$, W^∞ is the multiset in which every element is of infinite multiplicity: $\big(\forall a \in W\big)\big(W^\infty(a) = \infty\big)$.

Let $x, y \in \langle V, \mathbb{N}_\infty\rangle$ be two (possibly infinite) multisets over V. Then x is called a *submultiset* of y, written as $x \le y$, if and only if $\big(\forall a \in V\big)\big(x(a) \le y(a)\big)$. If $\big(\forall a \in V\big)\big(x(a) < y(a)\big)$ then x is called a strict submultiset of y. The sum of x and y, denoted by $x + y$ is defined in the following way: $\big(\forall a \in V\big)\big((x + y)(a) = x(a) + y(a)\big)$. The difference of x and y, denoted by $x - y$, is defined similarly: $\big(\forall a \in V\big)\big((x - y)(a) = x(a) - y(a)\big)$. The semantics of the symbol ∞ obey the usual rules: $\big(\forall n \in \mathbb{N}\big)\big(n \le \infty \wedge \infty + n = \infty - n = \infty\big)$. When talking about $x - y$, we assume that $y \in \langle V, \mathbb{N}\rangle$, i.e., the that subtracted multiset is finite.

If $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_m)$ are vectors of multisets over V, then the relation $X \le Y$ is defined as follows $\big(X \le Y\big) \Leftrightarrow \big(\forall i \in \mathbb{N}\big)\big(1 \le i \le m\big)\big(x_i \le y_i\big)$, i.e., $X \le Y$ if and only if each component of X is a submultiset of Y. Similarly, we define $X + Y$ and $X - Y$ in a component-wise way.

For further details on these topics see [2] and [6].

2.2 Clock-free P Systems

Clock-free P systems were originally introduced in [7]; an intuitive approach to working with the clock-free semantics was explored in [4].

A *clock-free* membrane system is defined by a tuple

$\Pi = (O, C, \mu, w_1, w_2, \cdots, w_m, R_1, R_2, \dots, R_m, i_0)$, where
O is a finite set of objects,
C is a finite set of catalysts, $C \in O$,
μ is a hierarchical structure of m membranes, bijectively labeled by $1, \dots, m$; 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.

The rules of a clock-free membrane system have the form $u \rightarrow v$, where $u \in O^+$ and $v \in (O \times Tar)^*$. In the case of non-cooperative rules, $u \in O$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$ are written in the following way: (a, t), $a \in O$, $t \in Tar$ and the target *here* is typically omitted. A rule associated with membrane i must only specify a label of the immediately inner membrane in a target indication in_j.

The rules are applied in a maximally parallel way: no further rule should be applicable to the idle objects. In the case of non-cooperative 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 $a \rightarrow u$, but these objects do not contribute to the result). Rules are non-deterministically chosen at each moment in time when a change occurs in the configuration of the P system. The process of choosing which rules should be applied does not take any time.

Intuitively, clock-free rule applications work in the following way. At the start of application, the multiset in the left-hand side of the rule is subtracted from the content of the corresponding region. When a rule application is complete, the multiset in the right-hand side of the rule is added to the corresponding region. The time between the start and the end of a rule application is a real value, may be different for different applications of the same rule, and there is impossible to know in advance.

For further definitions and details, see [7].

2.3 Networks of Cells

Networks of cells are a general framework for describing membrane systems with a static membrane structure. Networks of cells are formally described in depth in [3]. Intuitively, with this approach, membrane systems are considered as collections of interacting cells containing multisets of objects [8]. In this section we will only shortly expose the relevant considerations discussed in [3].

A *network of cells* of degree $n \geq 1$ is a tuple

$$\begin{aligned}
\mathit{\Pi} &= (n, V, w, \mathit{Inf}, R), \text{ where} \\
n &\quad \text{is the number of cells,} \\
V &\quad \text{is a finite alphabet,} \\
w &= (w_1, \ldots, w_n), w_i \in \langle V, \mathbb{N} \rangle, 1 \leq i \leq n, \text{is the initial content of cell } i, \\
\mathit{Inf} &= (\mathit{Inf}_1, \ldots, \mathit{Inf}_n),\ \mathit{Inf}_i = \{a \in V \mid w_i(a) = \infty\}, 1 \leq i \leq n, \\
R &\quad \text{is a finite set of interaction rules.}
\end{aligned}$$

According to the definition, the component Inf_i of the vector Inf contains the symbols which occur infinitely often in cell i. In most cases, only one cell, the environment, will contain symbols of infinite multiplicity.

The interaction rules in R have the form

$$(X \to Y, P, Q),$$

where $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ are vectors of finite multisets over V, i.e., $x_i, y_i \in \langle V, \mathbb{N} \rangle$, $1 \leq i \leq n$. Furthermore, $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$, with $p_i, q_i \in \langle V, \mathbb{N} \rangle$, $1 \leq i \leq n$. The vector P is sometimes called the permitting condition of the rule, while Q is sometimes referred to as the forbidding condition of the rule.

The following notation for the rule $(X \to Y, P, Q)$ is also used:

$$\big((x_1, 1) \ldots (x_n, n) \to (y_1, 1) \ldots (y_n, n),\ (p_1, 1) \ldots (p_n, n),\ (q_1, 1) \ldots (q_n, n)\big).$$

Whenever any of x_i, y_i, p_i, or q_i, $1 \leq i \leq n$, is empty, it may be omitted.

Initially, every cell contains $w_i \cup \mathit{Inf}_i^\infty$. An interaction rule rewrites the objects x_i from cells i, $1 \leq i \leq n$, into objects y_j in cells j, $1 \leq j \leq n$, if $\big(\forall i \in \mathbb{N}\big)\big(p_i \leq x_i \wedge \neg(q_i \leq x_i)\big)$, i.e., if x_i contains p_i and does not contain q_i, $1 \leq i \leq n$.

Note that the definition of the network of cells does not specify any structural relations between the cells. The reason is that in many P system models the structural organisation of membranes is mainly used to direct communication between the cells (as can be seen in [5]). In networks of cells, however, rules are allowed to modify any combinations of cells, thus removing the need for an explicit structure of cells as a means of organising communication.

A *configuration* of the network of cells $\mathit{\Pi}$ is an n-tuple of multisets over V: $C = (u'_1, \ldots, u'_n)$, in which $u'_i \in \langle V, \mathbb{N}_\infty \rangle$, $1 \leq i \leq n$. Configurations are often described by their finite parts only: $C^f = (u_1, \ldots, u_n)$, where $(u'_i = u_i \cup \mathit{Inf}_i^\infty) \wedge (u_i \cap \mathit{Inf}_i = \varnothing)$, $1 \leq i \leq n$ [3].

An interaction rule $r = (X \to Y, P, Q)$ is *eligible* in the configuration $C = (u_1, \ldots, u_n)$ if and only if the following condition is true:

$$\Big(\forall i \in \mathbb{N}\Big)\Big(1 \leq i \leq n\Big)\Big((p_i \leq u_i) \wedge \big((q_i = \varnothing) \vee \neg(q_i \leq u_i)\big) \wedge (x_i \leq u_i)\Big),$$

i.e., the corresponding cells contain all of the promoting multisets, do not contain the forbidding multisets and contain the corresponding multisets of the left-hand side of the rule. The set of rules eligible in configuration C of the network of cells $\mathit{\Pi}$ is denoted by $Eligible(\mathit{\Pi}, C)$.

Let $C = (u_1, \ldots, u_n)$ be a configuration of $\mathit{\Pi}$ and C^f be its finite part. Let $T \in \langle R, \mathbb{N} \rangle$, $supp(T) \subseteq Eligible(\mathit{\Pi}, C)$, be a finite multiset of eligible rules, $|T| = k$. Recall that every eligible rule has the following form: $r = (X \to Y, P, Q) \in supp(T)$. The algorithm which checks whether the multiset of rules T can be applied to the configuration C is described in Algorithm 1. The algorithm checks if the rules in T can *all at once* be applied to the configuration C. To perform the check, an attempt is made to remove the left-hand sides of the rules in T from C. If this is possible, the algorithm returns the multiset union of the left-hand sides of the rules in T, otherwise it returns $\varnothing$. See [3] for further details.

Algorithm 1 The marking algorithm

```
$T' \leftarrow T$
$Mark_0(\mathit{\Pi}, C, T) \leftarrow (\lambda, \ldots, \lambda)$ {empty vector of size $n$}
$i \leftarrow 1$
while $T' \neq \varnothing$ do
    $r = (X \to Y, P, Q) \leftarrow$ get $T'$
    $T' \leftarrow T' - \{r \to 1\}$
    if $X \leq C^f - Mark_{i-1}(\mathit{\Pi}, C, T)$ then
        $Mark_i(\mathit{\Pi}, C, T) \leftarrow Mark_{i-1}(\mathit{\Pi}, C, T) + X$
    else
        return $\varnothing$
    end if
    $i \leftarrow i + 1$
end while
return $Mark_k(\mathit{\Pi}, C, T)$
```

If, for the multiset of eligible rules $T \in \langle R, \mathbb{N} \rangle$, $supp(T) \subseteq Eligible(\mathit{\Pi}, C)$, the marking algorithm succeeds and $Mark(\mathit{\Pi}, C, T) \neq \varnothing$, the multiset T is called applicable to C. The set of all multisets applicable to the configuration C of $\mathit{\Pi}$ is denoted by $Appl(\mathit{\Pi}, C)$. The result of *applying* T to C is defined as follows:

$$Apply(\mathit{\Pi}, C, T) = C^f - \sum_{(X \to Y, P, Q) \in T} X + \sum_{(X \to Y, P, Q) \in T} Y,$$

or, equivalently,

$$Apply(\mathit{\Pi}, C, T) = (C^f - Mark(\mathit{\Pi}, C, T)) + \sum_{(X \to Y, P, Q) \in T} Y.$$

In plain words, $Apply(\mathit{\Pi}, C, T)$ is the configuration obtained from C by removing the left-hand sides of all rules in T and then adding the right-hand sides of all rules in T. Note that repeating entries of the same rule are treated independently both in $Mark$ and in *Apply*.

A *derivation mode* is a set of conditions applied to the set $Appl(\Pi, C)$ [3]. The maximally parallel derivation mode max is thus defined as follows:

$$Appl(\Pi, C, max) = \bigl\{T \in Appl(\Pi, C) \mid \bigl(\nexists T' \in Appl(\Pi, C)\bigr)\bigl(T' \supsetneq T\bigr)\bigr\}\,,$$

that is, $App(\Pi, C, max)$ contains those multisets of applicable rules which cannot be further maximised and remain applicable (which corresponds to the intuitive perception of the maximally parallel derivation mode).

Now fix a derivation mode ϑ (i.e. $\vartheta = max$). Consider two configurations C_1 and C_2 of Π. We say that $C_1 \Rightarrow_{(\Pi,\vartheta)} C_2$ if $\bigl(\exists T \in Appl(\Pi, C_1, \vartheta)\bigr)\bigl(C_2 = Apply(\Pi, C_1, T)\bigr)$, i.e., there exists an applicable multiset of rules T, valid under the derivation mode ϑ, with the property that applying T to C_1 yields C_2 [3]. In this case, C_2 is said to be the result of a transition step from C_1. When the network of cells and the derivation mode are clear from the context, the relation may be written as $\Rightarrow$. The reflexive and transitive closure of $\Rightarrow_{(\Pi,\vartheta)}$ is denoted by $\Rightarrow^*_{(\Pi,\vartheta)}$.

A configuration C of Π is said to be a halting configuration if C satisfies a certain halting condition. One of the most widely used halting conditions is $Appl(\Pi, C, \vartheta) = \varnothing$. Under this condition, C is a final configuration if there are no rules applicable to C under the derivation mode ϑ.

A *computation* of a network of cells Π under the derivation mode ϑ is the sequence of configurations $(C_i)_{i=0}^n$, where C_0 is the initial configuration, C_n is a halting configuration and $\bigl(\forall i \in \mathbb{N}\bigr)\bigl(0 \le i \le n-1\bigr)\bigl(C_i \Rightarrow_{(\Pi,\vartheta)} C_{i+1}\bigr)$. In plain words, a computation is a sequence of configurations which starts from the initial configuration and, by applications of multisets of rules valid under the derivation mode ϑ, reaches a halting configuration.

This section only contains a superficial overview of the corresponding material. Consider referring to [3] for further details on the formal framework for networks of cells.

3 Clock-free Networks of Cells

3.1 Preliminary Considerations

To understand and explore the concept of clock-freeness, we will start with an analysis of how a clock-free P system evolves. Consider the following sample clock-free P system:

$$\begin{aligned}
\Pi_0 &= (V, C, [\;\;]_1, w_1, R_1, 1), \text{ where}\\
V &= \{a, b, c, x, u\},\\
C &= \{x\},\\
w_1 &= a^3,\\
R_1 &= \{a \to b, b \to c, a \to u, a \to x, xu \to x\}.
\end{aligned}$$

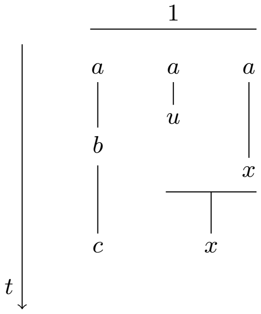


Fig. 1. The time diagram of a computation of Π_0

The time-diagram of a possible computation of Π_0 is shown in Figure 1. This diagram shows the possible evolution of individual symbols in the only region 1 of Π_0 as time progresses. In the initial state of the system, three different rules start consuming the three instances of a. At any moment sufficiently close to the initial state of the system, there are no symbols in region 1, because all of them have been consumed in the start of the three rule applications.

In this variant of evolution, the application of the rule $a \to u$ finishes first, producing an instance of u. Since there are no rules applicable to u, nothing happens at this time. The next rule application to finish is $a \to b$. At this moment, the contents of the only region of the system is bu. There is a rule $b \to c$, so, in accordance with the maximally parallel mode of evolution, this rule must be applied. The application of $b \to c$ therefore starts immediately after b has been produced and consumes the instance of b.

The next rule application to finalise is $a \to x$. As it can be seen from the diagram, at the moment when the first x is produced, the contents of region 1 will be ux. This renders the catalytic $xu \to x$ applicable, and so it is applied. In this variant of evolution, the applications of the rules $b \to c$ and $xu \to x$ finalise at exactly the same time. The system therefore halts with cx in its only region.

We explicitly remark that our choice of the variant of evolution is totally random. For example, the c may have not been produced at the same time with x. In fact, it could have been added to the system *before* u would be.

3.2 Clock-freeness: A Separate Concept

In this section we will tear apart the concept of a clock-free P system and focus on clock-freeness per se. The paper [7] introduces clock-free P system as a whole concept, without formally paying attention to the distinctive feature.

It turns out that clock-freeness cannot directly be described by any combination of the principal features of networks of cells considered in [3] (derivation mode, halting condition, goal of computations, interpretation of results). Indeed, the halting condition, the goal of computation, and the way to interpret results refer to the ending parts of the computation, while the derivation mode describes how to choose applicable multisets of rules. Clock-freeness, on the other hand, focuses on what happens after rules have been chosen, throughout the whole computation, not just in the closing phases.

The very special component of clock-freeness is that reaction times for rules are real numbers. This poses the question whether having real numbers in the model offers extended possibilities as compared to other models, where time is expressed as a natural number of steps. It turns out, however, that having arbitrary real numbers as reaction times is not at all defining. Indeed, the exact duration of a reaction is of no importance whatsoever. What deserves attention is only the relation of this value to other durations, i.e., we are only interested in knowing whether a multiset α was produced before, at the same time, of after multiset β. According to [7], we will consider the next configuration to be the instant description of the system when the output of a previously initiated reaction appears. Therefore, we consider the computation of a clock-free system to be a sequence of configurations, sampled at the moments when a rule application (or several rule applications) finalise. Observe that real numbers are relevant nowhere in this reasoning, since we always have a finite number of configurations in a computation.

Consider, for example, the computation shown in Figure 1. The sequence of configurations of this computation is the following:

$$(a^3), (u), (bu), (ux), (cx).$$

The computation starts with a^3 in the only region of the system. All three instances of a are consumed. Then a single u is produced. Later on, b is also added to the system. In this configuration, the rule $b \to c$ starts, so, when x is added to the system, there are no instances of b already. Finally, after both $b \to c$ and $ux \to x$ have finalised, the system stops in the halting configuration cx.

Observe again that we are not interested in reaction times themselves, but rather in the ordering of the moments of time when certain symbols were produced. Therefore, although the time intervals between configurations are some real numbers, we are free to discard this fact. Moreover, we can consider that a clock-free system transitions into the next configuration at every tick of a global clock, which brings us a huge step closer to the classic P system models. We lose absolutely nothing in this move because, as we have shown above, the exact duration of rules plays no role.

3.3 Formal Framework

Having done the preliminary consideration, we are now ready to introduce the clock-free mode of evolution into networks of cells. To be able to do that, we will extend the notion of a configuration.

Definition 1. *We call a* clock-free configuration *in a network of cells $\Pi = (n, V, w, Inf, R)$ the following construct:*

$$\begin{aligned}&C^* = (C, H),\ \textit{where}\\ &\ C = (u'_1, \dots, u'_n), u'_i \in \langle V, \mathbb{N}_\infty \rangle, 1 \le i \le n,\\ &\ H \in \langle R, \mathbb{N} \rangle.\end{aligned}$$

The initial *clock-free configuration of a network of cells is* $C_0^* = (C_0, \varnothing)$*, where* C_0 *describes the initial content of every cell in the network.*

A configuration C^* has two components. C describes the contents of the cells of the system in exactly the same way as a normal configuration of a network of cells does. R is a multiset of rules which are "still being applied". The exact semantics of this component will be revealed in the next paragraphs.

The way a transition step from a configuration $C_1^* = (C_1, H_1)$ into $C_2^* = (C_2, H_2)$ under the derivation mode ϑ is performed is described in Algorithm 2. Symbolically, what Algorithm 2 does is build C_2^* starting from C_1^* such that $C_1^* \Rightarrow_{(\Pi,\vartheta)} C_2^*$.

Algorithm 2 A clock-free transition step

```
1: $A \leftarrow \textbf{get}\ \mathit{Appl}(\Pi, C_1, \vartheta)$
2: $C_2 \leftarrow C_1 - \sum_{(X \to Y, P, Q) \in A} X$
3: $H' \leftarrow H_1 + A$
4: $F \leftarrow \textbf{get submultiset}\ H' \{\text{assure } F \neq \varnothing\}$
5: $C_2 \leftarrow C' + \sum_{(X \to Y, P, Q) \in F} Y$
6: $H_2 \leftarrow H' - F$
```

The transition from C_1^* starts by computing the set of multisets of applicable rules and picking one of them: A (line 1). The applications of the rules in A are started (line 2) and the rules themselves are added to the would-be second component of the new configuration (line 3). As remarked in the definition, the second component H_1 of the clock-free configuration $C_1^* = (C_1, H_1)$ is a multiset of rules, whose applications have not yet been finalised. The algorithm continues with picking an arbitrary nonempty submultiset of rules F to be finalised from H' (line 4). The right-hand sides of the rules in F are added to the system (line 5), while the rules themselves are removed from H' (line 6).

Observe that if instead of choosing an arbitrary submultiset $F \le H'$, Algorithm 2 would take $F = H'$, it would degenerate into the classic (non-clock-free) algorithm of computing the next configuration from the current one.

Further note that the requirement $F \neq \varnothing$ does not limit the domain of configurations this algorithm is applicable to, because H' could only be empty if the algorithm started from a halting configuration.

Now that we have described a clock-free configuration and the way transitions between configurations occur, we are ready to formally define the halting condition for a clock-free computation.

Definition 2. *A clock-free configuration $C^* = (C, H)$ of a network of cells Π evolving under the evolution mode ϑ is* halting *when all rules have finalised and there are no more applicable rules:*

$$H = \varnothing \bigwedge Appl(\Pi, C, \vartheta) = \varnothing.$$

This halting condition corresponds to the clock-free semantics as described in [7]. Obviously, just as with other variants of networks of cells, the predicate defining the halting condition can be defined in a different way.

In what follows, we will refer to networks of cells with clock-free configurations, operating according to Algorith 2, as to *clock-free networks of cells.*

3.4 Example of Clock-free Evolution (Algorithm 2)

We will now turn back to the example of a computation of the clock-free P system Π_0 shown in Figure 1. The computation starts with the initial configuration:

$$C_0^* = \Big((a^3), \varnothing\Big).$$

In this configuration the multiset of rules to apply is selected to be $\{(a \to b) \to 1, (a \to u) \to 1, (a \to x) \to 1\}$; the three instances of a are correspondingly removed. The rule $(a \to u)$ is immediately picked to be finalised, so the next configuration is

$$C_1^* = \Big((u), \big\{(a \to b) \to 1, (a \to x) \to 1\big\}\Big).$$

Since there are no rules which consume only u, nothing is applicable in C_2^*, so $A = \varnothing$ once again. This time $F = \{(a \to b) \to 1\}$, so the system transitions into

$$C_2^* = \Big((bu), \big\{(a \to x) \to 1\big\}\Big).$$

In this configuration the rule $b \to c$ becomes applicable, so its application must be started: the only b is removed and H' is correspondingly modified. The rule $a \to x$ is finalised ($F = \{(a \to x) \to 1\}$):

$$C_3^* = \Big((ux), \big\{(b \to c) \to 1\big\}\Big).$$

There is the rule $xu \to x$, so it must be started in this configuration. The algorithm then picks both rules which are "still being applied" (including the $xu \to x$, which has just been started) and finalises them:

$$C_4^* = \Big((cx), \varnothing\Big).$$

Since no rules are applicable and $H_4 = \varnothing$, the system has arrived at a halting configuration.

3.5 Suitability of the Formalisation

In this section we will discuss whether the formalism introduced and described in the previous section is compatible with the intuitive description of clock-freeness provided in [7]. In this paper we define clock-freeness on the foundation of networks of cells, while [7] starts from transitional P systems. To bridge the obvious gap, we will consider networks of cells with rules working in clock-free mode. This will bring a common ground to the (extended) definitions from [7] and the formal definitions suggested in this paper.

Definition 3. *A* *-network of cells *is a network of cells with rules operating in clock-free mode, in the sense of [7].*

According to the definition, rule applications in a *-network of cells last for a different real-valued time interval each. In this section we will only consider $\vartheta = max$ for both *-networks of cells and clock-free networks of cells, because clock-free P systems evolve under maximal derivation mode. The halting condition for *-networks of cells will be the condition that all rule applications have finalised and no more rules are applicable, while networks of cells with clock-free configurations will have the halting condition introduced in the previous sections. Since the goal of the computation and the way to interpret the result do not directly pertain to the subject of this section, we will consider these two parameters as having a certain well-defined value, the same for both kinds of analysed networks of cells.

Obviously, clock-free P systems as defined in [7] are a particular case of *-networks of cells.

Theorem 1 (Suitability of the formalism). *Consider a network of cells $\Pi = (n, V, w, Inf, R)$ and a finite sequence of clock-free configurations of Π: $\mathcal{C}^* = (C_i^*)_{i=0}^m$, with $C_0^* = (C_0, \varnothing)$ being an initial configuration and $C_m^* = (C_m, \varnothing)$ being a halting configuration. Let ${}^*\Pi = (n, V, w, Inf, R)$ be a *-network of cells. $\mathcal{C}^*$ is a clock-free computation if and only if the sequence of the first components $\mathcal{C} = (C_i)_{i=0}^m$ is a valid computation in ${}^*\Pi$.*

Proof. According to the corresponding definitions, C_0^* and C_n^* are valid clock-free initial and halting configurations in Π correspondingly if and only if C_0 and C_n are valid initial and halting configurations in ${}^*\Pi$ correspondingly. Obviously, the statement of the theorem holds for $C_0^* = C_m^*$, therefore we will focus on the cases when $m > 0$.

Consider C_0^* and C_1^* and suppose that $C_0^* \Rightarrow_{(\Pi,max)} C_1^*$. Then, in ${}^*\Pi$, we can consider the transition from C_0 to C_1 constructed in the following way: start the applications of the rules belonging the multiset A chosen in Algorithm 2, then consider that the rules collected into F as constructed in Algorithm 2 finalise their application at one and the same time. The moment these rules complete is the moment when C_1 will occur. Therefore, $C_0 \Rightarrow_{({}^*\Pi,max)} C_1$.

Now suppose that $C_0 \Rightarrow_{({}^*\Pi,max)} C_1$. This means that, in ${}^*\Pi$, some rule applications started in C_0, and some of these rules finished to result in configuration

C_1. We can therefore consider that in configuration C_0^*, Algorithm 2 chose to start the same rules as the ones which started in in ${}^*\Pi$ and then immediately finalised those which led to the occurrence of C_1 in ${}^*\Pi$. For C_1^* constructed in this way, the following is true: $C_0^* \Rightarrow_{(\Pi,max)} C_1^*$.

We have therefore proved that

$$\big(C_0^* \Rightarrow_{(\Pi,max)} C_1^*\big) \Leftrightarrow \big(C_0 \Rightarrow_{({}^*\Pi,max)} C_1\big).$$

Moreover, we have proved that H_1 contains those and only those rules which could have been started in ${}^*\Pi$ in configuration C_0 and might have not been finalised in transition to C_1. By repeating the same reasoning for any pair C_i^* and C_{i+1}^*, it is now possible to prove the statement of the theorem by induction.

4 Simulations of Clock-freeness

Now that we have formally defined clock-freeness, it is time to pose one of the most important questions: how "far away" are clock-free networks of cells from the traditional networks of cells? It turns out that it is very easy to simulate clock-freeness with traditional, static networks of cells, as they are formalised in [3].

Indeed, consider a clock-free network of cells $\Pi = (n, V, w, Inf, R)$ and a rule $r_i = (X \to Y, P, Q) \in R'$. According to clock-free semantics, an application of r_i is started in a configuration C_i^* of Π by removing its left-hand side from the system, and is finalised in a configuration C_j^* by adding its right-hand side to the system. Consider now an ordinary network of cells $\Pi' = (n, V', w, Inf, R')$, with R' and V' constructed in the following way:

$$\begin{aligned}
V' &= V \cup \big\{\xi_i \,\big|\, r_i = (X \to Y, P, Q) \in R\big\},\\
R' &= \Big\{\big(X \to (\xi_i, k^*), P, Q\big), \big(((\xi_i, k^*) \to (\xi_i, k^*), \bar\lambda, \bar\lambda\big),\\
&\qquad \big(((\xi_i, k^*) \to Y, \bar\lambda, \bar\lambda\big) \,\big|\, r_i = (X \to Y, P, Q) \in R\big\},\\
&\qquad 1 \le k^* \le |R|,
\end{aligned}$$

where $\bar\lambda$ is the vector of size n of empty multisets. The alphabet of Π' includes all symbols from the alphabet of Π, but also a ξ_i per each rule with index i.

For each rule in R, three rules are added to R'. When the rule r_i is applicable, instead of X being directly transformed into Y, X is initially rewritten into ξ_i, which is placed into the cell with index k^*. The choice of the index k^* is totally arbitrary, it may even be different for different rules. The symbol ξ_i can either reproduce itself or add Y to the system; either of these will unconditionally happen; the choice between the two options is nondeterministic.

We claim that Π' accurately simulates the evolution of Π. Indeed, a rule $(X \to (\xi_i q, k^*), P, Q) \in R'$ is applicable in Π' if and only if the corresponding rule $r_i = (X \to Y, P, Q) \in R$ is applicable. Rewriting X into ξ_i thus corresponds to removing the left-hand side of the rule from the system and adding it to H' (the operations performed in Algorithm 2). If, in a certain configuration, ξ_i is rewritten into itself, then, at this step, the rule r_i was not picked by Algorithm 2 to be finalised. The case when ξ_i is rewritten into Y corresponds to the scenario when Algorithm 2 picked rule r_i to finalise.

Observe now that if, in a certain configuration of Π', only the rules which rewrite ξ_i, $1 \le i \le |R|$, were chosen to be applied, then the system will arrive at exactly the same configuration at the next step. It is possible to detect such situations and cut off such computations. This however, does not make the following result significantly different on the overall.

Theorem 2 (Simulation of clock-freeness). *Consider a clock-free network of cells $\Pi = (n, V, w, Inf, R)$ and a sequence of clock-free configurations $\mathcal{C}^* = (C_i^*)_{i=0}^m$, with C_0^* being an initial configuration and C_m^* being a halting configuration. If $\mathcal{C}^*$ is a computation then it is possible to construct a computation $\mathcal{K} = (K_i)_{i=0}^l$ of an ordinary network of cells Π' so that there is a mapping $f : \mathcal{K} \to \mathcal{C}^*$ with the properties:*

1. *$f(K_i) = C_j^* = (C_j, H_j) \Leftrightarrow C_i \le K_i$ ($f(K_i) = C_i^*$ if and only if K_i contains at least all the symbols in K_i),*
2. *$\big(K_i \Rightarrow^*_{(\Pi', max)} K_j\big) \Leftrightarrow \big(f(K_i) \Rightarrow^*_{(\Pi, max)} f(K_j)\big)$ (f maps the binary relation $\Rightarrow^*_{(\Pi', max)}$ into $\Rightarrow^*_{(\Pi, max)}$ and vice versa),*
3. *$\big(\forall i\big)\big(\exists j\big)\big(f(K_j) = C_i^*\big)$ (f is surjective),*

where $1 \le i \le m$, $1 \le j \le l$.

Proof. According to the constructions in the previous paragraphs, to a clock-free network of cells Π, one can associate an ordinary network of cells Π' which simulates Π. The mapping f can thus be defined as follows: from the region i^* of K_i remove all instances of ξ_i, $1 \le i \le |R|$; this will be the first component of $f(K_i)$. The second component of $f(K_i)$ is obtained by starting with an empty multiset and adding an instance of rule r_i per each instance of symbol ξ_i, $1 \le i \le |R|$. The required properties of f follow from the constructions shown in this section.

5 Conclusion

In this paper we have torn apart the concept of clock-free P systems as defined in [7] and have separated clock-freeness as a stand-alone ingredient. We have formally defined this ingredient within the framework of networks of cells [3] and shown that this definition is consistent with the original concept. We have also shown that clock-freeness can be simulated with usual networks of cells in a quite straightforward way.

The fact that clock-freeness can be simulated so easily goes against the intuitive impression produced by clock-free P systems and poses the important question of how valuable this ingredient is. Indeed, it seems that almost any problem in clock-free systems can be equivalently formulated for the corresponding clocked systems.

We believe that clock-freeness is fairly important, though, because it is (intuitively) much closer to how processes take place in biological cells. The fact that clock-freeness is easy to simulate is thus beneficial and shows how it is possible to move closer to real life without sacrificing too much.

We remark, of course, that the chemical reactions taking place in the cell have been studied well enough to approximately predict their durations or, at least, compare them to other cellular processes in terms of speed. Clock-freeness, however, allows us to abstract away these details. Metaphorically put, an implementation of an operation in a clock-free network of cells (or clock-free P system) can survive changes in the physical implementation, because it does not depend on the durations of underlying chemical relations. This reasoning is of course hypothetical at the moment, but it may become practical quite soon.

In this paper we have used the term “clock-freeness” to denote the mode of evolution of a network of cells in which rule applications may last for an arbitrary long or short amount of time. The word “free” in the term “clock-free”, therefore, refers to a different concept than the same word in the term “time-free” [1]. However, the majority of clock-free P systems considered in [7] and, for example, [4], are in fact independent of the what the durations of rules are. It thus is possible to consider that the terms “clock-freeness” and “time-freeness” do have something in common. Observe that Theorem 2 allows translating the problem of independence of the durations of rules in clock-free networks of cells into the problem of confluence in regular networks of cells.

Acknowledgements

The author gratefully acknowledges Artiom Alhazov, PhD for the remarkably fruitful discussions during the work on [4], without which the present paper would not have happened in the first place.

References

1. A. Alhazov, M. Cavaliere: Evolution-communication P Systems: Time-Freeness. *Proc. Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo et al. eds.), 2005, 11–18.
2. J. Dassow, Gh. Păun: On the power of membrane computing. *Journal of Universal Computer Science*, 5 (1999), 33–49.
3. R. Freund, S. Verlan: A Formal Framework for Static (Tissue) P Systems. *8th International Workshop on Membrane Computing, WMC2007* (G. Eleftherakis et al., eds.), LNCS 4860, Springer, 2007.
4. S. Ivanov: Basic Concurrency Resolution in Clock-Free P Systems. *Proc. 12th International Conference on Membrane Computing* (M. Gheorghe et al., eds.), LNCS 7184, Springer, 2012, 226–242.
5. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
6. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 vols., Springer, Berlin, 1997.

7. D. Sburlan: Clock-free P Systems. *Pre-proc. Fifth Workshop in Membrane Computing* (G. Mauri, Gh. Paun, C. Zandron, eds.), 2004, 372–383.
8. F. Bernardini, M. Gheorghe, M. Margenstern, S. Verlan: Networks of Cells and Petri Nets. *Proc. 5th Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo et al., eds.), 3 2007, 3–62.
9. The P Systems Web Page: `http://ppage.psystems.eu`.

On the Simulations of Evolution-Communication P Systems with Energy without Antiport Rules for GPUs

Richelle Ann B. Juayong[1], Francis George C. Cabarle[1], Henry N. Adorna[1], Miguel A. Martínez–del–Amor[2]

[1] Algorithms & Complexity Lab
Department of Computer Science
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines
E-mail: rbjuayong@up.edu.ph, fccabarle@up.edu.ph, hnadorna@dcs.upd.edu.ph

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es

Summary. In this report, we present our initial proposal on simulating computations on a restricted variant of Evolution-Communication P system with energy (ECPe system) which will then be implemented in Graphics Processing Units (GPUs). This ECPe systems variant prohibits the use of antiport rules for communication. Several possible levels of parallelizations for simulating ECPe systems computations on GPUs are emphasized. Our work is based on a localized matrix representation for the mentioned variant given in a previous literature. Our proposal employs a methodology for forward computing also discussed in the said literature.

Key words: Membrane computing, Parallel computing, GPU computing

1 Introduction

Evolution-Communication P systems with energy (ECPe systems) [1] is a variant of P systems introduced in 2009 to initiate a framework for communication complexity. It originates from Evolution-Communication P (ECP) systems [10], a hybrid of two well-investigated variants, Transition P systems [9] and P systems with Symport and Antiport rules [11]. The difference between ECPe and ECP systems is the presence of a special object called 'energy' in the former, which can be produced through evolution rules and consumed in communication rules. One crucial restriction for ECPe systems includes the use of energy for each communication rule. Thus, no object can be communicated without using some 'quanta' of energy. Moreover, upon being delivered to a receiving region, the energy used in a communication does not pass through any membrane. In this manner, it is said that the energy used in the process of communication is being 'lost'.

Several recent works have introduced the concept of representing certain P system variants and their computations as matrices and vector-matrix operations, respectively. In particular, variants known as Spiking Neural P (SNP) systems and their matrix representations were introduced in [5] whereas matrix representations for ECPe systems were given in [6]. Aside from creating a 'convenient' and relatively compact way of describing the systems and their computations, the matrix representations add additional ease to their simulation and implementation in parallel hardware. Vector and matrix operations are highly parallelizable and can be efficiently implemented in parallel hardware, including Graphics Processing Units (GPUs). GPUs are massively parallel hardware not like current generation CPUs. Using their respective matrix representations, SNP systems have been successfully implemented in GPUs in [2] and more recently in [3]. The intention of this current work is to continue such trend i.e. to provide our methodology on how to implement ECPe systems computations (using the vector-matrix representations) on parallel hardware, in particular GPUs.

2 Evolution-Communication P Systems with Energy

2.1 Formal Definition of ECPe systems

Before we proceed, we note that the readers are assumed to be familiar with the fundamentals of formal language theory and membrane computing [9].

A relatively new variant of Evolution-Communication P systems [10] has been introduced in [1] to evaluate communication that is dependent on some energy produced from evolution rules. A special object e is introduced to the system to represent a quantum of energy. We use the definition for EC P system with energy (ECPe system) from [1].

Definition 1. *An EC P system with energy is a construct of the form*

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

where:

(i) m pertains to the total number of membranes;
(ii) O is the alphabet of objects;
(iii) μ is the membrane structure which can be denoted by a set of paired square brackets with labels. We say that membrane i is the *parent membrane* of a membrane j, denoted $parent(j)$, if the paired square brackets representing

membrane j is located inside the paired square brackets representing membrane i, i.e. $[_i \ldots [_j\ ]_j]_i$. Reversely, we say that membrane j is a *child membrane* of membrane i, denoted $j \in children(i)$ where $children(i)$ refers to the set of membranes contained in membrane i. The relation of parent and child membrane becomes more apparent when we represent the membrane structure as a tree. Since order does not matter in our model, there can be multiple trees (isomorphic with respect to children of a node), each corresponding to the same membrane structure representation.

(iv) $w_1, \ldots, w_m$ are strings over O^* where w_i denotes the multiset of object present in the region bounded by membrane i.

(v) $R_1, \ldots, R_m$ are sets of evolution rules, each associated with a region delimited by a membrane in μ;

- An evolution rule is of the form $a \to v$ where $a \in O$, $v \in (O \cup \{e\})^*$. In the event that this type of rule is applied, the object a transforms into a multiset of objects v in the next time step. Through evolution rules, object e can be produced, but e should never be in the initial configuration and object e is not allowed to evolve.

(vi) $R'_1, \ldots, R'_m$ are sets of communication rules, each associated with a membrane in μ; A communication rule can either be a symport or an antiport rule:

- A symport rule can be of the form (ae^i, in) or (ae^i, out), where $a \in O$, $i \geq 1$. By using this rule, i copy of e objects are consumed to transport object a inside (denoted by in) or outside (denoted by out) the membrane where the rule is defined. To consume copies of object e means that upon completion of the transportation of object a, the occurrences of e are lost, they do not pass from a region to another one.
- An antiport rule is of the form $(ae^i, out; be^j, in)$ where $a, b \in O$ and $i, j \geq 1$. By using this rule, we know that there exists an object a in the region immediately outside the membrane where the rule is declared, and an object b inside the region bounded by the membrane. In the application of this rule, object a and object b are swapped using i and j copies of object e in the different regions, respectively. As in symport rules, the copies of object e are lost after the application.

We say that a communication rule has a *sending* and *receiving* region. For a rule $r \in R'_i$ associated with an in label, its receiving region is region i and its sending region is the $parent(i)$. On the other hand, the sending and receiving regions are reversed for a rule $r \in R'_i$ associated with an out label. For an antiport rule $r \in R'_i$, region i and $parent(i)$ are both sending and receiving region. Also, note that no communication can be applied without the utilization of object e.

(vii) $i_{out} \in \{0, 1, \ldots, m\}$ is the output membrane. If $i_{out} = 0$, this means that the environment shall be the placeholder of the output.

Rules are applied in a nondeterministic, maximally parallel manner. Nondeterminism, in this case, has the following meaning: when there are more than two evolution rules that can be applied to an object, the system will randomly choose the rule to be applied for each copy of the object. The system assumes a universal clock for simultaneous processing of membranes; all applicable rules have to be applied to all possible objects at the same time. The behavior of maximally parallel application of rule requires that all object that can evolve (or be transferred) should evolve (or be transferred).

Note that there is a one-to-one mapping between region and membrane, however, strictly, region refers to the area delimited by a membrane. A configuration at any time i, denoted by C_i, is the state of the system; it consists of the membrane structure and the multiset of objects within each membrane. A transition from C_i to C_{i+1} through nondeterministic and maximally parallel manner of rule application can be denoted as $C_i \Rightarrow C_{i+1}$. A series of transition is said to be a computation and can be denoted as $C_i \Rightarrow^* C_j$ where $i < j$. Computation succeeds when the system halts; this occurs when the system reaches a configuration wherein none of the rules can be applied. This configuration is called a halting configuration. If there is no halting configuration—that is, if the system does not halt—computation fails, because the system did not produce any output. Output can either be in the form of objects sent outside the skin, the outermost membrane, or objects sent into an output membrane.

We let $N(\Pi)$ be the set of numbers generated by a given ECPe system Π.

2.2 An Example

To show how ECPe system works, we shall give an example of an ECPe system with two membranes adapted from [1]:

$$\Pi = (\{a, \#\}, e, [_1[_2]_2]_1, a^2\#, \lambda, \{r_{11} : a \rightarrow aa, r_{12} : a \rightarrow ee\}, \emptyset,$$
$$\{r_{21} : \# \rightarrow \#\}, \{r'_{21} : (ae, in), r'_{22} : (\#e, in)\}, 2)$$

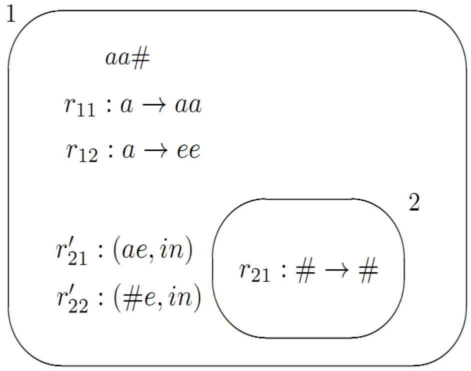

1
$aa\#$
$r_{11} : a \rightarrow aa$
$r_{12} : a \rightarrow ee$
2
$r'_{21} : (ae, in)$
$r'_{22} : (\#e, in)$
$r_{21} : \# \rightarrow \#$

Fig. 1. Graphical representation of an ECPe system Π where $N(\Pi) = \{2(n+1)|n \geq 0\}$ from [1].

A graphical illustration of Π is shown in Figure 1. Its output is $N(\Pi) = \{2(n+1)|n \geq 0\}$. The computation to generate this proceeds as follows:

Initially, we can use either rule r_{11} or r_{12} in order to consume the copies of object a in membrane 1. At any time, we can use both r_{11} and r_{12} to evolve copies of object a. For every copy of a, we produce either two copies of object e or another two copies of object a, therefore, we are always assured that multiplicity of object a in region 1 is even, as well as the multiplicity of object e. Note that upon introduction of object e in the system, it should immediately be used (in the next step) to transport copies of object a in region 2, otherwise, it will be used to transport the trap symbol # in region 2 using rule r'_{22} leading the system to a never ending computation due to rule r_{21}. For a computation to halt, rule r'_{21} should be the last rule to be applied. Since copies of object e transporting copies of object a in region 2 is always even, we are assured that the multiplicity of objects (copies of object a) in region 2 is also even. The minimum value 2 is produced when we use both rule r_{11} and r_{12} in configuration C_1. In the next step, we use two applications of rule r'_{21}. This will cause the system to halt.

2.3 Representation and Methodology for Forward Computing

In this section, we relate how computations in ECPe systems without antiport rules can be performed in a localized manner. As will be shown, when we do not allow antiport rules, membranes can compute more independently. This representation has been used in [4] to answer the problem of computing backward and forward. We shall relate the methodology for the latter. By computing forward, the problem is to find the next configurations that can be yielded in one computational step given a current configuration.

Let $h \in \{0, 1, 2, \ldots, m\}$ where region 0 refers to the region located outside the skin, the outermost membrane. The following notations and definitions are adopted from [4] and used in the remaining parts of this section.

- Let $IO(r, h)$ be the set of objects in region h involved in a rule r.
- Let $TO(r, h)$ be the set of objects in region h that trigger a rule r.
- The set of rules $IR(h) = R_h \cup R'_h \cup (\bigcup_{h' \in children(h)} R'_{h'})$ represents the set of rules that directly influences the content of region h at any time of a computation. An object α in region h at any time $i \geq 0$ may either be produced by an evolution rule, transported from neighboring region to region h (or vice versa), or simply carried over. In the first scenario, it is by definition that the rule that produced object α must be in R_h. A neighboring region may either be a region delimited by $parent(h)$ or regions delimited by membranes in $children(h)$. In the first case, the rules for communication are in R'_h while in the second case, the rules for communication must be in one of $R'_{h'}$ where $h' \in children(h)$.
- The set $PO(h) = \{\alpha|\alpha \text{ appeared in } w_h\} \cup (\bigcup_{r \in IR(h)} IO(r, h))$ represents the set of objects (including special object e) that may possibly occur in region h at any time of a computation. Originally, the objects that surely exist in the region are the elements present in w_h. In order to create a copy of an object

α, object α must either be produced or transported in region h through rules in $IR(h)$.

- The set $TR(h) = \{r | TO(r, h) \neq \emptyset\}$ corresponds to the set of rules that contribute to the decrease of objects in region h. In order to activate rules belonging to such set, there must be a trigger object that may either be consumed or be used for transportation

In order to represent configuration and rule application in terms of vectors, and represent effect of a rule in each region using a matrix, the concept of total order must be utilized. We note that for all the vector (and matrix) representation constructed in the remaining parts of this section, there is a need to define a *total order* $\langle p_1, p_2, ...\rangle$ (so that p_i is considered the i^{th} element in a defined set) over the elements involved in the column for vectors (rows and columns for matrices). As can be observed, this is used so that elements are uniquely identified by their positions in the order to where they belong to and to assure that the position of elements are correct during the vector-matrix operation.

Definition 2. ***Configuration Vector for each Region*** h
A configuration vector $\mathbf{C}_{i,h}$ *is a vector whose length is* $|PO(h)|$*. The vector* $\mathbf{C}_{i,h}(\alpha)$ *refers to the multiplicity of object* α *in region* h *at configuration* C_i*.*

Definition 3. ***Application Vector for each Region*** h
An application vector $\mathbf{a}_{i,h}$ *is a vector whose length is* $|R(h)|$*. The vector* $\mathbf{a}_{i,h}(r)$ *refers to the number of application of rule* r *specifically in region* h *during the transition* $C_{i-1} \Rightarrow C_i$*.*

Definition 4. ***Transition Matrix for each Region*** h
A transition matrix $M_{\Pi_{ECPe},h}$ *is a matrix whose dimension is* $|R(h)| \times |PO(h)|$*. The matrix* $M_{\Pi_{ECPe},h}(r, \alpha)$ *returns the number of consumed or produced object* α *in region* h *upon single application of rule* r*. The consumed objects have negative values while the produced objects are positive. If object* α *in region* h *is not used in rule* r*, then its value is zero.*

Given application vector $a_{i,h}$ representing a maximal set of rule applications applied in a configuration C_{i-1} to achieve configuration C_i, the paper [4] showed that a transition $\mathbf{C}_{i-1} \rightarrow \mathbf{C_i}$ can be represented by performing

$$\mathbf{C}_{i,h} = \mathbf{C}_{i-1,h} + \mathbf{a}_{i,h}.M_{\Pi,h} \tag{1}$$

for each region h provided that if h and h' are the sender and receiver regions corresponding to a communication rule $r' \in IR(h) \cap IR(h')$, then $\mathbf{a}_{i,h}(r') = \mathbf{a}_{i,h'}(r')$.

Illustrating Localized Computation

To illustrate localized computation, we represent a possible transition $C_0 \Rightarrow C_1$ by showing the effect of applying rule r_{11} and rule r_{12} once on the initial configuration of the example presented in Section 2.2. Since $PO(0) = IR(0) = \emptyset$, the participation of the environment is not needed in any part of the computation.

For Region 1

At the onset, we can impose a total order $\langle a, \#, e\rangle$ over $PO(1)$ and total order $\langle r_{11}, r_{12}, r'_{21}, r'_{22}\rangle$ over $IR(1)$. The initial configuration will be represented by the configuration vector $\mathbf{C}_{0,1}$ where

$$\mathbf{C}_{0,1} = \begin{pmatrix} 2 \; 1 \; 0 \end{pmatrix}$$

and the representation for single application of both rules r_{11} and r_{12} will be given by application vector $\mathbf{a}_{1,1}$ where

$$\mathbf{a}_{1,1} = \begin{pmatrix} 1 \; 1 \; 0 \; 0 \end{pmatrix}$$

Applying Equation (1) with the transition matrix $M_{\Pi_{ECPe},1}$ containing the values shown below:

$$M_{\Pi_{ECPe},1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 2 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{pmatrix}$$

will yield the configuration vector $\mathbf{C}_{1,1}$

$$\mathbf{C}_{1,1} = \begin{pmatrix} 2 \; 1 \; 2 \end{pmatrix}$$

which means that in the next configuration, there will be two copies of both object a and the special object e and a single copy of the trap symbol # in region 1.

For Region 2

For region 2, we impose total order $\langle a, \#\rangle$ over $PO(2)$ and total order $\langle r_{21}, r'_{21}, r'_{22}\rangle$ over $IR(2)$. Since initially, no objects are present in region 2 and the rules involved in the transition $C_0 \Rightarrow C_1$ are not in $IR(2)$, the configuration vector $\mathbf{C}_{0,2}$ and the application vector $\mathbf{a}_{1,2}$ will all contain zero values. We now show the transition matrix $M_{\Pi_{ECPe},2}$

$$M_{\Pi_{ECPe},2} = \begin{pmatrix} 0 \; 0 \\ 1 \; 0 \\ 0 \; 1 \end{pmatrix}$$

Since application vector $\mathbf{a}_{1,2}$ is a zero vector, the configuration vector $\mathbf{C}_{1,2}$ remains also a zero vector.

Notice that all the declared communication rules influence the multiplicity of objects in both region 1 and region 2. However, region 1 contains negative values because it acts as a sending region while region 2 have non-negative values since it acts as a receiving region. Also, matrix $M_{\Pi_{ECPe},2}$ shows that the special object e can never reach region 2.

Forward Computing in ECPe systems without Antiport Rules

Shown below is a methodology for forward computing shown in [4].

1. Categorize all possible objects in $PO(h)$ for all region h.

First, all $\alpha \in PO(h)$ are categorized for a certain region h. These categories are:

- Category 1: Evolution Trigger
  Object α is an evolution trigger if there exists $r \in R_h$ such that $TO(r,h) = \{\alpha\}$.
- Category 2: Communication Trigger Only
  Object α belongs in this category if there does not exist $r \in R_h$ such that $TO(r,h) = \{\alpha\}$ but there exists $r' \in IR(h)$ such that $\alpha \in TO(r',h)$.
- Category 3: Not a Trigger
  Object α is neither in Category 1 nor in Category 2.

2. Construct identity rules for objects in Category 2 and 3 for all region h.

For each $\alpha \in PO(h)$ that falls under one of Category 2 and Category 3, an identity rule $\alpha \to \alpha$ is added. All these rules shall be contained in a set labelled $R_{add,h}$. Also, a list of $\alpha' \in PO(h) - \{e\}$ that fall under Category 2 is maintained, the list shall be labelled $List_{cat_2}$ and sorted $List_{cat_2}$ in increasing order of energy requirement for transport.

3. Construct Trigger Matrix $TM_{\Pi_{ECPe},h}$ for all region h

The defined rules represented in the rows of $TM_{\Pi_{ECPe},h}$ are the rules that contribute to the decrease of multiplicity of objects in region h. These rules are represented in the set $TR(h)$. The additional rules from $R_{add,h}$ are represented in the rows as well. The set of objects represented in the columns of $TM_{\Pi_{ECPe},h}$ is $PO(h)$. Therefore, $TM_{\Pi_{ECPe},h}$ has dimensions $|TR(h) \cup R_{add,h}| \times |PO(h)|$. $TM_{\Pi_{ECPe},h}(r,\alpha)$ returns the multiplicity of α in region h needed to activate a single application of rule r.

4. Set the dimension of the vector of unknowns (also called extended application vector) $\mathbf{a'}_{i,h}$ for all region h

The length of $\mathbf{a'}_{i,h}$ is $|TR(h) \cup R_{add,h}|$.

5. Solve system of linear equation

Find all solutions to the equation

$$\mathbf{a'}_{i,h}.TM_{\Pi_{ECPe},h} = \mathbf{C}_{i-1,h} \qquad (2)$$

Since elements of vector $\mathbf{a'}_{i,h}$ pertain to number of application of rules, these elements must be natural numbers. The value $\mathbf{a'}_{i,h}(r)$ can be interpreted as either the number of application of each rule $r \in TR(h)$ or how many object α is unevolved or unmoved (if $(r : \alpha \to \alpha) \in R_{add,h}$). Note that $TR(h)$ and $R_{add,h}$ are disjoint sets.

6. Filter solutions in Step 5

For each region h, if $List_{cat_2} \neq \emptyset$, scan the sorted $List_{cat_2}$ and find out the first object, labelled $\alpha_{cat_2,min}$, falling under Category 2 whose corresponding identity rule application is non-zero, i.e. $\mathbf{a'}_{i,h}(\alpha_{cat_2,min} \rightarrow \alpha_{cat_2,min}) > 0$. Since $List_{cat_2}$ is sorted increasingly according to transport energy requirement, the object $\alpha_{cat_2,min}$ has the minimum energy required for communication. Let its corresponding energy be labelled $energy(\alpha_{cat_2,min})$. Solutions are filtered in step 5 by adding, for each region h with a non-empty $List_{cat_2}$, the inequality below:

$$\mathbf{a'}_{i,h}(e \rightarrow e) < energy(\alpha_{cat_2,min}) \qquad (3)$$

7. Finding $\mathbf{a}_{i,h}$

Upon finding values for $\mathbf{a'}_{i,h}$ in all region h, all identity rules $r' \in R_{add,h}$ are omitted. The values of an application vector $\mathbf{a}_{i,h}$ are filled through the equation

$$\mathbf{a}_{i,h}(r) = \mathbf{a'}_{i,h}(r),\ r \in R_h \qquad (4)$$

For every communication rule $r \in IR(r, h') \cap IR(r, h'')$,

$$\mathbf{a}_{i,h'}(r) = \mathbf{a}_{i,h''}(r) = \mathbf{a'}_{i,h''}(r) \qquad (5)$$

where region h'' is the sending region of communication rule r.

An Illustration

We illustrate how we can compute forward in ECPe systems without antiport by using the ECPe system given in Section 2.2. We maintain the total orders $\langle a, \#, e\rangle$ over elements of $PO(1)$, $\langle a, \#\rangle$ over elements of $PO(2)$, $\langle r_{11}, r_{12}, Add_{11}, Add_{12}\rangle$ over elements of $ER(1) \cup R_{add,1}$ and $\langle r_{21}, r'_{21}, r'_{22}, Add_{21}\rangle$ over elements of $ER(2) \cup R_{add,2}$. Thus, our vectors are:

$$\mathbf{C}_{i-1,1} = \begin{pmatrix} 2 & 1 & 2 \end{pmatrix} \qquad \mathbf{C}_{i-1,2} = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Step 1

For region 1, object a belong to Category 1, object # and special object e belong to Category 2 while no objects belong to Category 3. On the other hand, objects # and a in region 2 belong to Category 1 and Category 3, resp.

Step 2

The additional identity rules per region are given below.

$$R_{add,1} = \{Add_{11} : \# \rightarrow \#, Add_{12} : e \rightarrow e\}$$
$$R_{add,2} = \{Add_{21} : a \rightarrow a\}$$

Since only object # is in category 2, $List_{cat_2}$ for region 1 is composed of only a single element #.

Step 3 and 4

The trigger matrix for both region 1 and 2 are shown below

$$TM_{\Pi_{ECPe},1} = \begin{pmatrix} 1\,0\,0 \\ 1\,0\,0 \\ 1\,0\,1 \\ 0\,1\,1 \\ 0\,1\,0 \\ 0\,0\,1 \end{pmatrix} \qquad TM_{\Pi_{ECPe},2} = \begin{pmatrix} 0\,1 \\ 1\,0 \end{pmatrix}$$

The extended application vectors $\mathbf{a'}_{i,1}$ and $\mathbf{a'}_{i,2}$ representing the vector of unknowns has the same index as that of the rows of their corresponding effect matrix.

Step 5

The resulting system of linear equations achieved from Equation (2) for region 1 and 2 is given below:

$$\begin{gathered} \mathbf{a'}_{i-1,1}(r_{11}) + \mathbf{a'}_{i-1,1}(r_{12}) + \mathbf{a'}_{i-1,1}(r'_{21}) = 2 \\ \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{11}) = 1 \\ \mathbf{a'}_{i-1,1}(r'_{21}) + \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{12}) = 2 \\ \mathbf{a'}_{i-1,2}(r_{21}) = 0 \\ \mathbf{a'}_{i-1,2}(Add_{21}) = 1 \end{gathered}$$

As can be traced, there are 11 possible extended application vectors for region 1 while there exists a unique extended application vector for region 2. Shown below is extended application vector for region 2:

$$\mathbf{a'}_{i-1,2} = \begin{pmatrix} 0\,1 \end{pmatrix}$$

Step 6

The additional inequality in region 1 requires that:

$$\mathbf{a'}_{i,1}(Add_{12}) < 1$$

for cases where the trap object # remain. Thus, these solutions are possible:

$$\mathbf{a}_{i,1} = \begin{pmatrix} 0\,0\,2\,0\,1\,0 \end{pmatrix} \qquad \mathbf{a}_{i,1} = \begin{pmatrix} 1\,0\,1\,1\,0\,0 \end{pmatrix}$$

$$\mathbf{a}_{i,1} = \begin{pmatrix} 0\,1\,1\,1\,0\,0 \end{pmatrix}$$

For cases where the trap object does not remain, the following solutions are also possible:

$$\mathbf{a}_{i,1} = \begin{pmatrix} 1\,1\,0\,1\,0\,1 \end{pmatrix} \qquad \mathbf{a}_{i,1} = \begin{pmatrix} 0\,2\,0\,1\,0\,1 \end{pmatrix}$$

$$\mathbf{a}_{i,1} = \begin{pmatrix} 2\,0\,0\,1\,0\,1 \end{pmatrix}$$

After step 6, the 11 solutions in step 5 were reduced to six.

Step 7

Performing Equation (4) and Equation (5), below are the possible application vector combinations:

$$\begin{aligned}
&\textit{Solution } 1: && \mathbf{a}_{i,1} = \begin{pmatrix} 0\,0\,2\,0 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\,2\,0 \end{pmatrix} \\
&\textit{Solution } 2: && \mathbf{a}_{i,1} = \begin{pmatrix} 1\,0\,1\,1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\,1\,1 \end{pmatrix} \\
&\textit{Solution } 3: && \mathbf{a}_{i,1} = \begin{pmatrix} 0\,1\,1\,1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\,1\,1 \end{pmatrix} \\
&\textit{Solution } 4: && \mathbf{a}_{i,1} = \begin{pmatrix} 1\,1\,0\,1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\,0\,1 \end{pmatrix} \\
&\textit{Solution } 5: && \mathbf{a}_{i,1} = \begin{pmatrix} 0\,2\,0\,1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\,0\,1 \end{pmatrix} \\
&\textit{Solution } 6: && \mathbf{a}_{i,1} = \begin{pmatrix} 2\,0\,0\,1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\,0\,1 \end{pmatrix}
\end{aligned}$$

The corresponding configuration vectors for each solution is as follows:

$$\begin{aligned}
&\textit{Solution } 1: && \mathbf{C}_{i,1} = \begin{pmatrix} 0\,1\,0 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 3\,0 \end{pmatrix} \\
&\textit{Solution } 2: && \mathbf{C}_{i,1} = \begin{pmatrix} 2\,0\,0 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 2\,1 \end{pmatrix} \\
&\textit{Solution } 3: && \mathbf{C}_{i,1} = \begin{pmatrix} 0\,0\,2 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 2\,1 \end{pmatrix} \\
&\textit{Solution } 4: && \mathbf{C}_{i,1} = \begin{pmatrix} 2\,0\,3 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 1\,1 \end{pmatrix} \\
&\textit{Solution } 5: && \mathbf{C}_{i,1} = \begin{pmatrix} 0\,0\,5 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 1\,1 \end{pmatrix} \\
&\textit{Solution } 6: && \mathbf{C}_{i,1} = \begin{pmatrix} 4\,0\,1 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 1\,1 \end{pmatrix}
\end{aligned}$$

2.4 A Sequential Implementation of Computation on ECPe Systems without Antiport using Initial Matrix Representation

Given the representation and algorithm for forward computing presented in Section 2.3, we were able to do a sequential implementation of computation on ECPe systems without antiport using the C programming language.

The system starts with reading two input files containing information for finding valid application vectors and determining the next configuration vector given a currently examined configuration vector. The following are the names of the input files:

- File *trans_file.txt* which contains the information needed for transitioning from one configuration to the next.
- File *forwComp_file.txt* which contains the information needed to find valid configuration vector/s given a current configuration vector.

A discussion about the format for the specified files is given in the appendix. The files will be used to initialize the necessary variables and pointers needed for the simulation. The system has two output files representing the tree-structure of the configuration history. The following are the name of the output files:

- File *conf.txt* which contains a list of configuration.
- File *conf_index.txt* which contains the index of the configuration in the tree structure.

Presented in Figure 2 is the flowchart of how the program works. Upon reading input files and loading variables and pointers needed for computing, the initial configuration is placed in the *conf.txt* and the associated index 1 is placed in *conf_index.txt*. Afterwards, the system will enter a loop for determining the configurations generated by a currently examined configuration. The examination of configuration will be executed in order of their position in the file. Given two configuration C and C' where the position of C precedes C', then configuration C will be examined first before configuration C'. Examining configuration shall halt only when the system reaches two stopping criterion:

- Upon achieving a pre-specified upper bound on the number of iterations
- Upon reaching a state where there are no more configurations to examine.

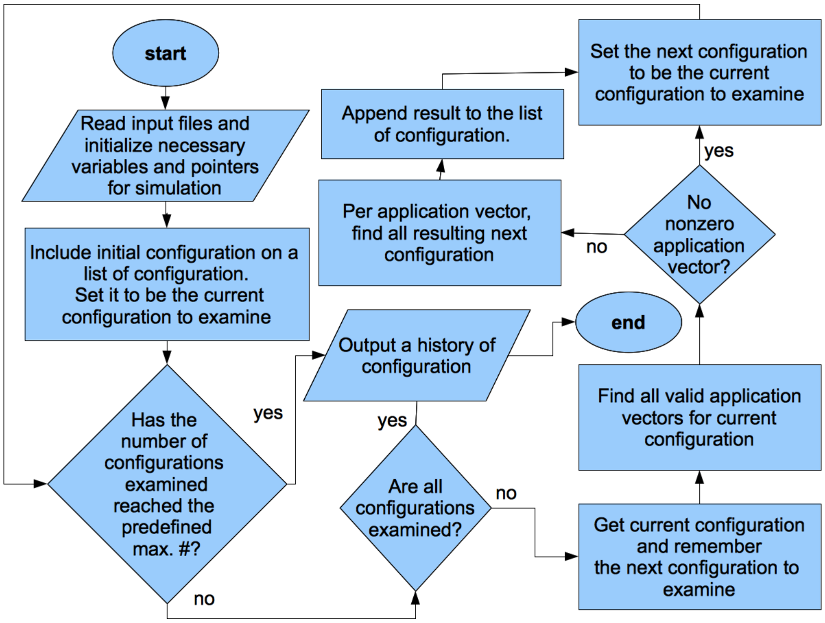

start
Read input files and initialize necessary variables and pointers for simulation
Include initial configuration on a list of configuration. Set it to be the current configuration to examine
Has the number of configurations examined reached the predefined max. #?
yes
no
Output a history of configuration
end
yes
Are all configurations examined?
no
Get current configuration and remember the next configuration to examine
Find all valid application vectors for current configuration
No nonzero application vector?
yes
no
Set the next configuration to be the current configuration to examine
Append result to the list of configuration.
Per application vector, find all resulting next configuration

Fig. 2. An overview of our sequential implementation for ECPe systems without antiport in the C programming language.

For every loop, a configuration in the file is examined by first determining all valid application vectors which is applicable to the currently examined configuration. In finding a valid application vector, we use the concept of localized representation and extended application vectors, and follow the steps in Section 2.3 for forward computing. If the only application vector applicable is the zero vector which means no more rule combination can be applied to the current configuration, it will proceed to the next configuration to examine. Each non-zero application vector is used to generate the next set of configurations. Afterwards, output files *conf.txt* and *conf_index.txt* will be updated to account all the newly generated next configuration vectors. Note that in this system, we have not yet implemented the methodology to detect repeating configuration.

On Generating Extended Valid Application Vectors and Filtering

The goal of step 5 in Section 2.3 is the generation of all possible extended application vectors $a'_{i,h}$ satisfying equation (2). In our sequential implementation, we achieve this by examining each equation resulting from the corresponding and equivalent system of linear equation. We now study the characteristics of the resulting system by using the example presented for forward computing.

Shown below are the equations yield from Equation (2) of region 1 for the example in Section 2.2, also shown in Section 2.3.

$$\beta_1 : \mathbf{a'}_{i-1,1}(r_{11}) + \mathbf{a'}_{i-1,1}(r_{12}) + \mathbf{a'}_{i-1,1}(r'_{21}) = 2$$
$$\beta_2 : \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{11}) = 1$$
$$\beta_3 : \mathbf{a'}_{i-1,1}(r'_{21}) + \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{12}) = 2$$

It can be observed that each equation in the resulting system represents an *object condition*; the object referring to possible objects that may enter an examined region. Also, for sending regions, an equation for *energy condition* (β_3) must also be present in a resulting system. In the general case, each variable (representing rule application of a certain communication rule) in the energy equation is present in exactly one other object condition. This object is the communication trigger that will be communicated upon activation of the rule represented by the said variable. For example, the variable $\mathbf{a'}_{i-1,1}(r'_{21})$ is present in both β_1 and β_3. The same goes for variable $\mathbf{a'}_{i-1,1}(r'_{22})$ which is present in both β_2 and β_3.

Other than such type of variables, no more variables can be present in more than one equation. Moreover, while the coefficients of the terms in the energy equation can contain any positive integer, the coefficients of the terms for non-energy condition will always be one (due to the restriction of noncooperative rule format). Moreover, the set of rules $r \in TR(h)$ are all represented by the union of all rules (variables) represented in the non-energy conditions without the identity rules.

Given such observation, possible vectors $a'_{i,h}$ are determined by first solving the condition posed for energy. Since there can be multiple solution for rules involving energy (the rules include the identity rule for energy since it is of Category 2), we shall determine the possible extended applications vectors resulting from a valid energy solution. In the linear equation shown above, we first on determining solutions for β_3. The possible *energy solutions* are

$$(1, 1, 0), (1, 0, 1), (0, 1, 1), (2, 0, 0), (0, 2, 0), (0, 0, 2),$$

where in each vector, the first element corresponds to value of $\mathbf{a'}_{i-1,1}(r'_{21})$, the second corresponds to the value of $\mathbf{a'}_{i-1,1}(r'_{22})$ and the third element is for the energy identity rule $\mathbf{a'}_{i-1,1}(Add_{12})$. For each solution, we then copy the rule application to the associated object to communicate. Afterwards, the rule application is transferred to the right-hand side of the equation, i.e. subtracted from the current count of the corresponding communicated object. As an example, the resulting modified object condition β'_1 caused by the value of communication rule$\mathbf{a'}_{i-1,1}(r'_{21})$ in the first energy solution will be

$$\beta'_1 : \mathbf{a'}_{i-1,1}(r_{11}) + \mathbf{a'}_{i-1,1}(r_{12}) = 1$$

If the resulting count is negative, then, the resulting energy solution cannot be applied. Therefore, no vector $a'_{i,h}$ can be generated given such negative result. This filtering on energy solution is evident in applying energy solution (0,2,0) on β_2. Upon subtracting the resulting rule application from the right-hand side of the corresponding communicated object condition, the resulting object conditions can be analyzed one at a time.

Upon realizing solutions for each object condition, step 6 of the forward computing methodology can already be executed per object condition. Identity rules for category 2 objects can be further checked to execute the filtering part, done in Step 6 of the methodology in Section 2.3 to see if the number of category 2 objects remaining in the region can be allowed to remain (that is, the case doesn't validate the rule that all objects that can evolve or be communicated must do so). Otherwise, the object solution will be dropped. Note that while step 6 in Section 2.3 evaluates first all extended application vectors before this step, we perform this step per object condition since the identity rules for any category 2 objects can only be present in the corresponding category 2 object equation. Preferably, the order of analyzing category 2 objects follow the sorted list $List_{cat_2}$ so that if an object solution is not valid, it can terminate immediately at the first unsatisfied category 2 object.

The resulting value of each filtered variable set per solution can be combined, one solution from each object, and each combined list constitutes one extended application vector. To illustrate this, we examine the possible extended application vectors that can be yield from energy solution $(1, 1, 0)$. For β_1 condition, the object solutions are $(0, 1)$ and $(1, 0)$ where the first element of the said vectors correspond to $\mathbf{a'}_{i-1,1}(r_{11})$ and the second, to $\mathbf{a'}_{i-1,1}(r_{12})$. For β_2 condition, the object solution only assigns the value 1 to $\mathbf{a'}_{i-1,1}(Add_{11})$. Therefore, for energy solution (1,1,0), the corresponding extended application vectors yielded are

$$\mathbf{a}_{i,1} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \qquad \mathbf{a}_{i,1} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

3 Simulator design and implementation

In this section, we relay how we can employ GPUs to parallelize the task of finding all possible object solutions. NVIDIA introduced the Compute Unified Device

Architecture (CUDA) in 2007 [7]. CUDA is a software and hardware architecture for general purpose computations in NVIDIA's GPUs [7]. CUDA extends high-level languages such as C to allow programmers to easily create software that will be executed in parallel, avoiding low-level graphics and hardware primitives [12].

GPUs introduce increased performance speedups over CPU only implementations with linear algebra computations (among other types of computations) because of the GPU architecture. The common CPU architectures are composed of transistors which are divided into different blocks to perform the basic tasks of CPUs (general computation): control, caching, DRAM, and ALU (arithmetic and logic). In contrast, only a fraction of the CPU's transistors allocated for control and caching are used by GPUs, since far more transistors are used for ALU [7] (see Figure 3 for an illustration). This architectural difference is a very distinct and significant reason why GPUs offer large performance increases over CPU only implementation of parallel code working on large amounts of input data. However if the problem to be solved cannot be organized in a data parallel form (a task performing computations on data need not depend heavily on other task's results) then the performance of GPUs over CPUs will not be fully utilized.

Code written for CUDA can be split up into multiple threads within multiple thread blocks, each contained within a grid of (thread) blocks. These grids belong to a single device or GPU. Each device has multiple cores, each capable of running its own threads. Each core in the device is able to run a set of threads.A thread block is assigned to each multiprocessor, where each processor is made up of several cores [7, 12]. A function known as a *kernel function* is one that is called from the host or CPU but executed in the device. Using kernel functions, the programmer can specify the GPU resources: the layout of the threads (from one to three dimensions) in a thread block, and the thread blocks (from one to two dimensions) in a grid. Table 1 shows the resources of current CUDA enabled NVIDIA GPUs.

GPU resources	**Values**
Global memory	Up to 4GB
Max number of threads per dimension (x, y, z)	$(1024, 1024, 64)$
Max number of thread blocks per grid (x, y, z)	$(65535, 65535, 65535)$

Table 1. Typical resources for CUDA enabled Fermi architecture GPUs (from [7, 12]) .

On Parallelizing Transitions

Another apparent possibility in order to simulate the parallel computations of ECPe systems (as well as capitalize on their representations as matrices) on GPUs, we can have initially at least two levels of parallelism: the first level is the computation of Equation (1) in parallel by threads in a block; the second level involves the computation of all the possible next configurations given a current configuration, so that each block in a grid of thread blocks performs this level.

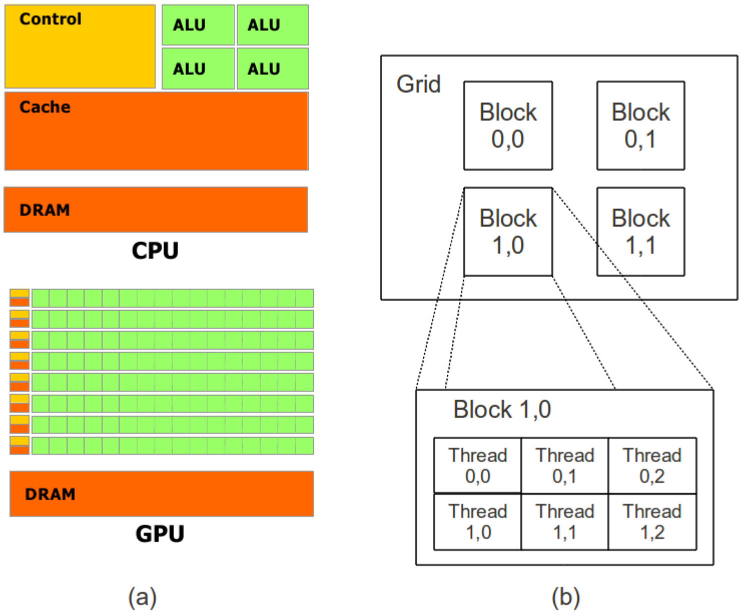


Fig. 3. (a) Common transistor allocation of CPUs and GPUs (b) Computing unit hierarchy of GPUs, from [12].

The first level is highly parallelizable since vector-matrix multiplication and vector addition are highly data independent. Each thread can multiply a vector to one column of the matrix, thus performing $a_{i,h} \cdot M_{\Pi,h}$. Each thread sums the products then adds these to another vector, performing the addition of $C_{i-1,h}$. For the second level, if there are q number of $a_{i,h}$'s and hence q number of next configurations, then q blocks will perform Equation (1) q times.

Because of the physical limitations of current NVIDIA GPUs, no more than 1024 threads per block are allowed for Fermi architecture GPUs so that at most matrices of at most 1024 columns can be simulated in a block. In Fermi GPUs, the maximum number of allowable thread blocks in a grid is 65535 (See Table 1) so q is currently upper bounded by this value. Another simulation consideration, aside from the computing units (threads, thread blocks) is the relatively more limited memory of current GPUs compared to CPUs. In this case, storing all q number of application vectors (each of which are of length $|R(h)|$) and the q number of next configurations (each of which are of length $|PO(h)|$) resulting from those application vectors must fit into the GPU's global memory.

On Generating Object Solutions

Given an examined energy solution, we check each object condition where each variable corresponding to a communication rule have already been determined (via the examined energy solution) and the value at the right-hand side of a communicated object has already been updated. As can observed, this problem is reduced to an integer partition problem where, given an object equation $\alpha_1+\alpha_2+...+\alpha_k=n$, we need to find vector containing $\alpha_i \in \mathbb{N}$'s, i.e. $(\alpha_1,\alpha_2,....,\alpha_k)$.

To generate solutions for non-energy object equation, we first obtain a *lexicographic order* of partition for the value n. In [13], given $X=(x_1,x_2,...,x_{k'})$ and $Y=(y_1,y_2,...,y_{k''})$, X precedes Y lexicographically if and only if for some $j \geq 1$, $x_i \geq y_i$ when $i<j$, and x_j precedes y_j. As an example, partitions of 5 in lexicographic order are: $11111, 2111, 311, 221, 311, 32, 41, 5$.

Each resulting partition will be padded with zeroes accordingly so that the partition can be represented in a k-dimensional vector. Each partition will be assigned to a thread. Each thread will be responsible for the generation of distinct permutation of the k-dimensional vector representing the partition. The union of solutions generated by each partition corresponds to the set of object solutions for a certain examined object. Since the filtering step of the forward methodology, as explained in Section 2.4, can be done per object condition, this step can also be performed within the current threads.

The idea of parallelization in Section 3 may also be used for generating energy solutions of the form $c_1\alpha_1+c_2\alpha_2+...+c_k\alpha_k=n$, where a lexicographic order on the partitions of n is first accomplished. Again, each resulting partition will then be padded with zeroes accordingly. Upon assigning each partition to a thread, and generating distinct permutation of a partition, each vector representating a permutation can be equated with the vector $(c_1\alpha_1,c_2\alpha_2,...,c_k\alpha_k)$, afterwhich the corresponding value for the $\alpha_i's$ can be obtained.

4 Conclusions and future work

In this report, we were able to describe a sequential implementation of a forward computing methodology for ECPe systems without antiport rules. We also were able to show how we extend this sequential work to employ GPUs for parallelizing some key areas in the implementation procedures. We were also able to show our proposed ideas for parallelizing other parts of the code.

As future work, we would like to implement our proposed ideas for parallelization and test the efficiency of the resulting implementation. Moreover, we hope to improve these ideas to better capitalize the parallelizability of vector-matrix representations of the computations on GPUs. As part of our future works, we also would like to extend the methodology for forward computing to apply to a general ECPe systems where antiport rules are allowed. The difficulty in allowing such communication rules are influenced by the implication that a region can be both a sender and receiver region. Thus, antiport rules need to maintain relationship between adjacent regions at crucial parts of the forward computing methodology. The more tricky part is the action done when considering objects that are unmoved due to antiport rule. The antiport rule increases the number of possible cases dictating why a category 2 object can remain in a certain region.

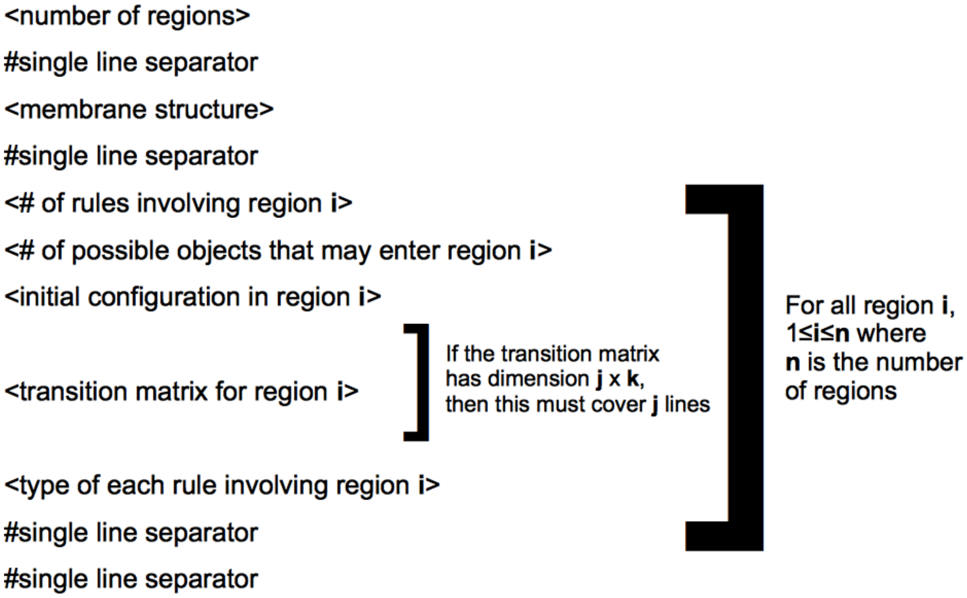
5 Acknowledgments

R.B. Juayong and F.G.C. Cabarle are partially supported by the DOST-ERDT scholarship program. H. Adorna is funded by the DOST-ERDT research grant and the Alexan professorial chair of the UP Diliman Department of Computer Science, University of the Philippines Diliman. M.A. Martínez–del–Amor is supported by "Proyecto de Excelencia con Investigador de Reconocida Valía" of the "Junta de Andalucía" under grant P08-TIC04200, and by the project TIN2009–13192 of the "Ministerio de Ciencia e Innovación" of Spain, both co-financed by FEDER funds.

References

1. H. Adorna, Gh. Păun, M. Pérez-Jiménez : On Communication Complexity in Evolution-Communication P systems, *Romanian Journal of Information Science and Technology, Vol. 13 No. 2 pp. 113-130, 2010*
2. F.G.C. Cabarle, H. Adorna, M.A. Martínez-del-Amor: A Spiking Neural P system simulator based on CUDA, M. Gheorghe et al. (Eds.), *12th Int'l Conference on Membrane Computing 2011*, revised and selected papers, LNCS vol. 7184, pp. 87-103. Springer-Verlag, 2012.
3. F.G.C. Cabarle, H. Adorna, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez: Improving GPU Simulations of Spiking Neural P Systems, (to appear) *Romanian Journal of Information Science and Technology*, 2012.
4. R.A. Juayong, H. Adorna: Computing on Evolution-Communication P systems with Energy Using Symport Only, *Workshop on Computation: Theory and Practice 2011 (WCTP 2011), UP Diliman NISMED auditorium.*
5. X. Zeng, H. Adorna, M. A. Martínez-del-Amor, L. Pan, M. Pérez-Jiménez : Matrix Representation of Spiking Neural P Systems, *Membrane Computing: Lecture Notes in Computer Science, Volume 6501/2011, 377-391, 2011*
6. R.A. Juayong, H. Adorna: A Matrix Representation for Computations in Evolution-Communication P Systems with Energy, *Proc. of Philippine Computing Science Congress, Naga, Camarines Sur, Philippines, March 3-4, 2011*
7. Kirk D., Hwu W., *Programming Massively Parallel Processors: A Hands On Approach*, 1st ed. MA, USA, Morgan Kaufmann, 2010.
8. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Computing Backwards with P systems, *WMC*10, Curtea de Argeş, Romania, (2009), 282-295.
9. Gh. Păun: Introduction to Membrane Computing. In: Gabriel Ciobanu, Mario J. Pérez-Jiménez and Gheorghe Păun, eds: Applications of Membrane Computing, Natural Computing Series. Springer, pp.142. (2006)
10. M. Cavaliere: Evolution-communication P systems. *Membrane Computing. Proc. WMC 2002, Curtea de Argeş* (Gh. Păun et al., eds.), LNCS 2597, Springer, Berlin, 134-145. (2003)
11. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing, 20, 3, 295-306*, (2002)
12. NVIDIA corporation, *"NVIDIA CUDA C programming guide"*, version 3.2, CA, USA, 2010.
13. A. Zoghbi, I. Stojmenović: Fast algorithms for generating integer partitions, *International Journal of Computer Mathematics*, Vol. 70, No. 2., pp. 319-332, (1998)

Appendix A: On File Formats for Implementation of ECPe system in C



<number of regions>

#single line separator

<membrane structure>

#single line separator

<# of rules involving region **i**>

<# of possible objects that may enter region **i**>

<initial configuration in region **i**>

<transition matrix for region **i**>

If the transition matrix has dimension **j** x **k**, then this must cover **j** lines

<type of each rule involving region **i**>

#single line separator

For all region **i**, 1≤**i**≤**n** where **n** is the number of regions

#single line separator

Fig. 4. Format for file *trans_file.txt* for the sequential implementation of ECPe system in C.

Shown in Figure 4 above is the format for input file *trans_file.txt*. This file contains the information required to find a succeeding configuration given a current one. First, the number of regions must be specified. If there will be a case where the environment will be needed during the computation, the value of this parameter must be the number of membranes incremented by one in order to account for the environment. The membrane structure is represented by paired square brackets as typical representation of a membrane structure. Note that for either a closed or open square bracket, the symbol must be followed by a numeral to indicate the label of the membrane. For the initial configuration, there is a need to follow a total order as discussed in Section 2.3. Therefore, the initial configuration is simply a initial configuration vector where each cell contains the number of copies of a certain object at the start of the computation. The separator for the cells will be the space symbol. The expected transition matrix has dimensions following the previously specified number of rules involving a specific region and the number of possible objects that may enter the region. The rows will be separated by newline, whereas, the columns are separated by spaces. A rule type can either be an evolution rule in which case the symbol to type will be ’e’, whereas a communication rule can have one of symbol ’s’ and ’r’. The symbol ’s’ represents that the communication rule uses the specific region as a sender, while the symbol ’r’ uses the specific region as a receiver.

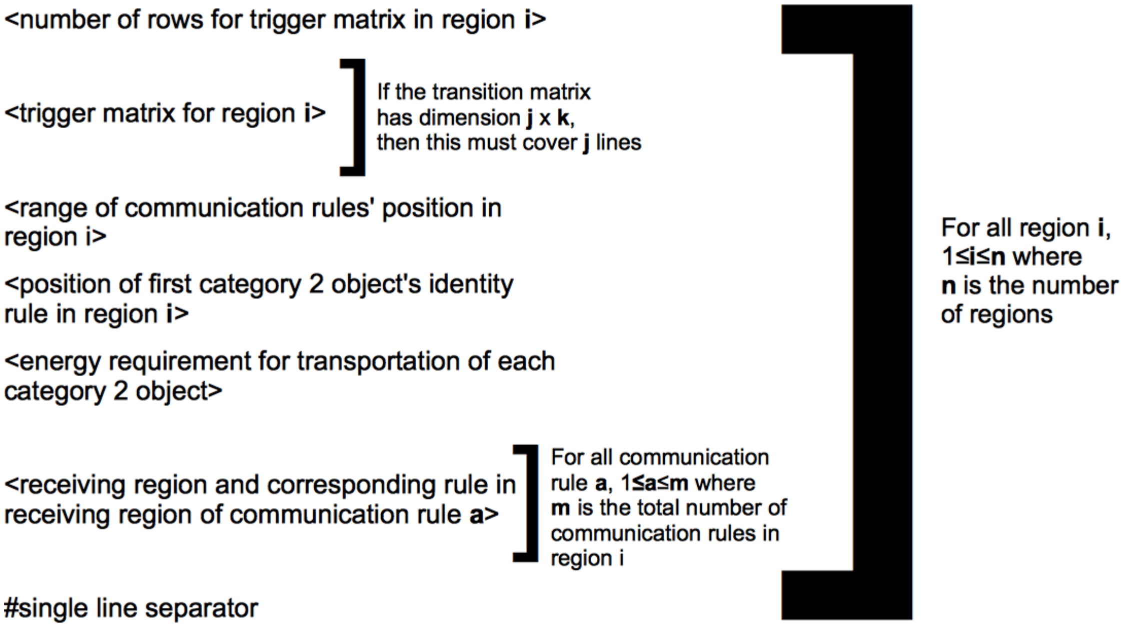
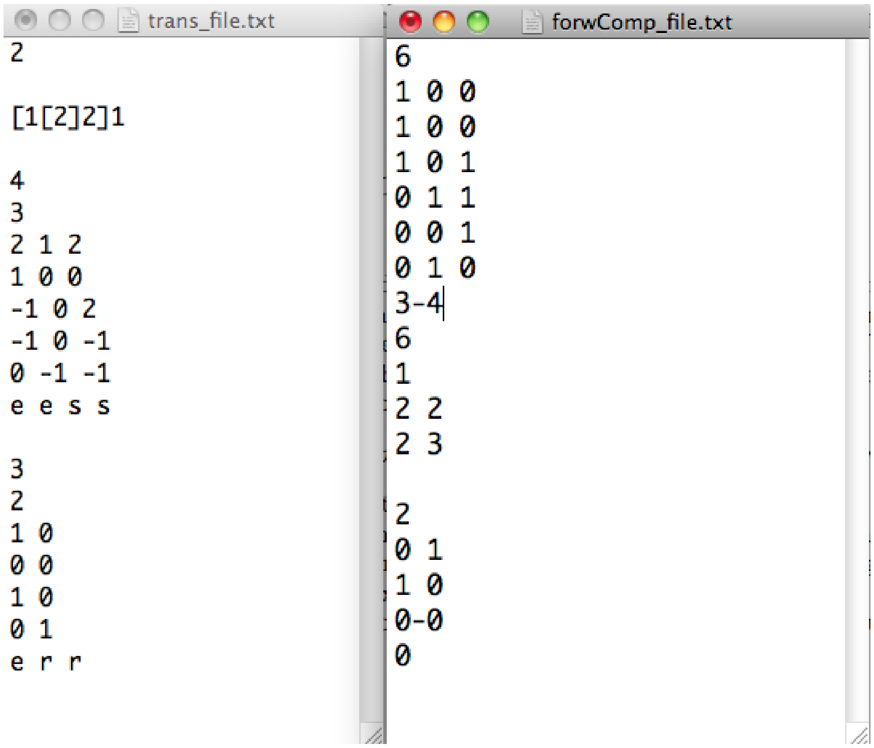

<number of rows for trigger matrix in region **i**>
<trigger matrix for region **i**>
If the transition matrix has dimension **j** x **k**, then this must cover **j** lines
<range of communication rules' position in region i>
<position of first category 2 object's identity rule in region **i**>
<energy requirement for transportation of each category 2 object>
<receiving region and corresponding rule in receiving region of communication rule **a**>
For all communication rule **a**, **1≤a≤m** where **m** is the total number of communication rules in region i
#single line separator
For all region **i**, 1≤**i**≤**n** where **n** is the number of regions

Fig. 5. Format for file *forwComp_file.txt* for the sequential implementation of ECPe system in C.

Figure 5, on the other hand, illustrates the format for the input file *forwComp_file.txt*. This file contains the information required to find all valid application vectors from a given configuration vector. First, a trigger matrix needs to be indicated. Again, there is a need to impose a total order over the rules. In this case, there will be a specific setup for the rules in the trigger matrix wherein, it is required that evolution rules must be specified first before the communication rules. After the existing rules associated with the region, the identity rule for energy (e) should proceed after. The identity rules for the other category 2 objects will follow after the energy’s identity rule. After indicating the trigger matrix, there is a need to define the range of the communication rules in the specified region. To indicate this:



```
2

[1[2]2]1

4
3
2 1 2
1 0 0
-1 0 2
-1 0 -1
0 -1 -1
e e s s

3
2
1 0
0 0
1 0
0 1
e r r
```

```
6
1 0 0
1 0 0
1 0 1
0 1 1
0 0 1
0 1 0
3-4
6
1
2 2
2 3

2
0 1
1 0
0-0
0
```

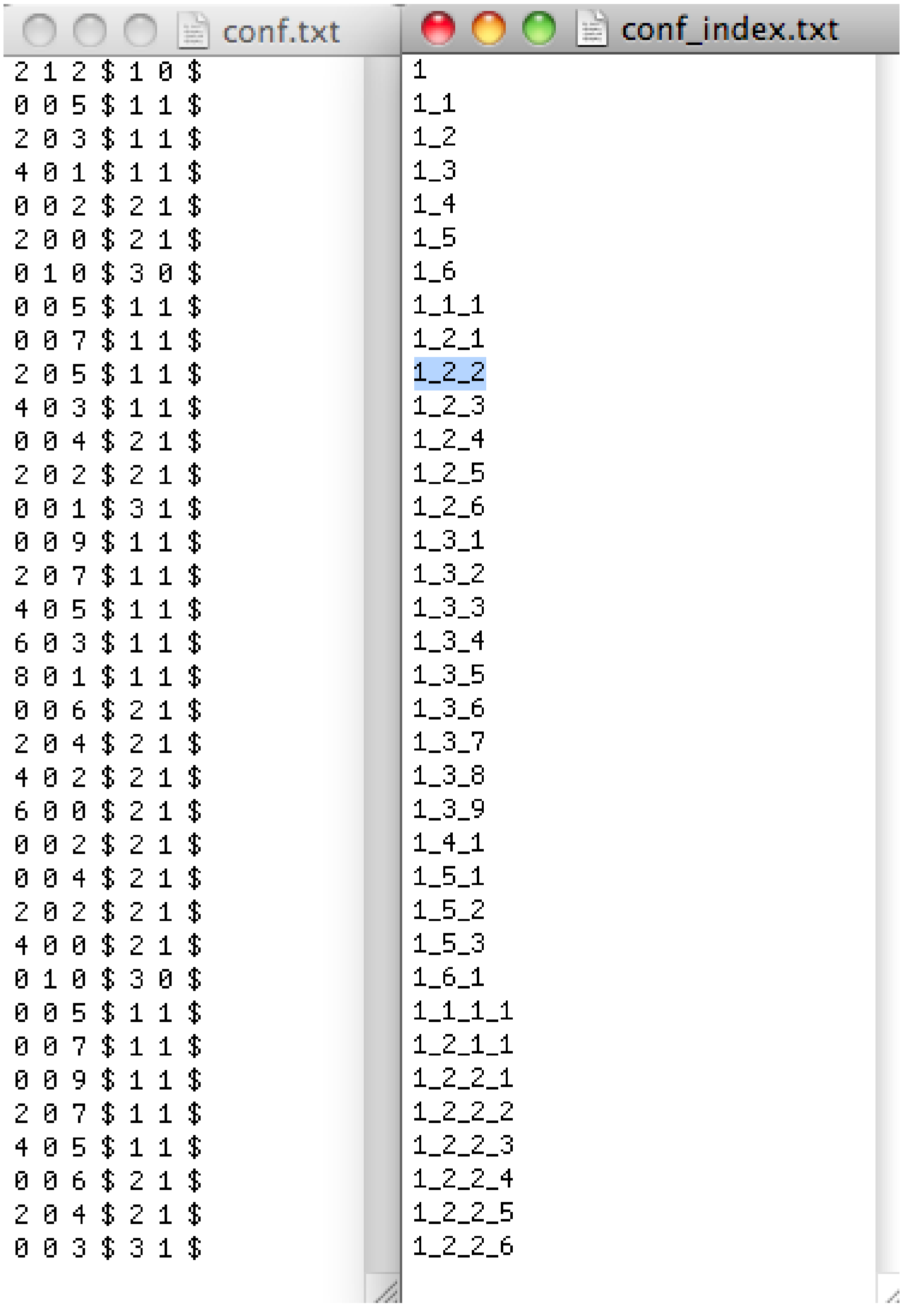
Fig. 6. Input files for the example given in Section 2.2

<position of starting communication rule>-<position of last communication rule>

where the positions will be based on the imposed total order. The total order will also be used in the next set of input which entails specifying the minimal energy requirement needed to transport each category 2 object. Since they only be enumerated in a single line, the separator of energy per object will be the space symbol. Following after this will be the enumeration of the details needed in the receiving region for each communication rule. To indicate this:

<position of receiving region> <position of partner rule in receiving region>

where this specifications per communication rule will be separated by newlines. Note that only a single space symbol separates the position of the receiving region with the position of the corresponding rule in the receiving region. In case no communication rules exist in the trigger matrix, there is no need to fill up this part. Moreover, the range of communication rules will be 0-0 and the value of the succeeding line will be 0. Shown in Figure 6 are the input files for the example given in Section 2.2 whose total order for involved rules and possible objects follows the order given in Section 2.3 except that we swap the position of the last and second



```
2 1 2 $ 1 0 $
0 0 5 $ 1 1 $
2 0 3 $ 1 1 $
4 0 1 $ 1 1 $
0 0 2 $ 2 1 $
2 0 0 $ 2 1 $
0 1 0 $ 3 0 $
0 0 5 $ 1 1 $
0 0 7 $ 1 1 $
2 0 5 $ 1 1 $
4 0 3 $ 1 1 $
0 0 4 $ 2 1 $
2 0 2 $ 2 1 $
0 0 1 $ 3 1 $
0 0 9 $ 1 1 $
2 0 7 $ 1 1 $
4 0 5 $ 1 1 $
6 0 3 $ 1 1 $
8 0 1 $ 1 1 $
0 0 6 $ 2 1 $
2 0 4 $ 2 1 $
4 0 2 $ 2 1 $
6 0 0 $ 2 1 $
0 0 2 $ 2 1 $
0 0 4 $ 2 1 $
2 0 2 $ 2 1 $
4 0 0 $ 2 1 $
0 1 0 $ 3 0 $
0 0 5 $ 1 1 $
0 0 7 $ 1 1 $
0 0 9 $ 1 1 $
2 0 7 $ 1 1 $
4 0 5 $ 1 1 $
0 0 6 $ 2 1 $
2 0 4 $ 2 1 $
0 0 3 $ 3 1 $
```

```
1
1_1
1_2
1_3
1_4
1_5
1_6
1_1_1
1_2_1
1_2_2
1_2_3
1_2_4
1_2_5
1_2_6
1_3_1
1_3_2
1_3_3
1_3_4
1_3_5
1_3_6
1_3_7
1_3_8
1_3_9
1_4_1
1_5_1
1_5_2
1_5_3
1_6_1
1_1_1_1
1_2_1_1
1_2_2_1
1_2_2_2
1_2_2_3
1_2_2_4
1_2_2_5
1_2_2_6
```

Fig. 7. Sample output files for the example given in Section 2.2

to the last identity rule in the first region to follow the required format for total order on rules involved in the corresponding trigger matrix.

For the output, there will be two files, namely *conf.txt* and *conf_index.txt*, that shall represent an ECPe systems configuration tree of computations. The former consists a list of configurations (not yet necesarily unique) where each system configuration is separated by a newline. The system configuration is composed of configuration vectors local to each region that are juxtaposed together. The elements of the vectors are separated by an individual space whereas dollar sign ($) separates each local configurations. The initial configuration will be the first configuration in the file.

The output file *conf_index.txt* stores the indices of the configurations in file *conf.txt* in order to remember the association between configurations. Given an index in this file, the configuration associated with the index is the configuration in *conf.txt* which has the same line position as the line of the index. For example, the initial configuration is located at the first line in the file $conf.txt$, its corresponding first line in $conf\_index.txt$ is the index 1. Since we can represent computation as a tree, the initial configuration with index 1 is the root node configuration of the tree. To determine the configurations that sprung from the initial configuration i.e. the children of the root node, the indices must have the prefix '1_'. The parent configuration which generates a configuration can be tracked by removing the last underscore in the index associated with the configuration, along with the number that appears after the said underscore. A path from the initial configuration to any configuration C can be traced by retrieving the associated configuration starting from index 1 (which, as mentioned, represents the initial configuration) to the configuration C following the precedence imposed by the indices. Shown in Figure 7 are the output files for the example given in Section 2.2 whose total order for objects follows the order given in Section 2.3 and where the maximum number of configurations to examine is set to 10.

Towards an Integrated Approach for Model Simulation, Property Extraction and Verification of P Systems

Raluca Lefticaru[1], Florentin Ipate[1], Luis Valencia Cabrera[2],
Adrian Ţurcanu[1], Cristina Tudose[1], Marian Gheorghe[3],
Mario de J. Pérez-Jiménez[2], Ionuţ Mihai Niculescu[1], and Ciprian Dragomir[3]

[1]Department of Mathematics and Computer Science, University of Piteşti
Str. Târgu din Vale 1, 110040, Piteşti, Romania
{name.surname}@upit.ro

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

[3]Department of Computer Science, University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
{m.gheorghe,c.dragomir}@sheffield.ac.uk

Summary. This paper presents an integrated approach for model simulation, property extraction and formal verification of P systems, illustrated on a tissue P system with active membranes solving the 3-colouring problem. The paper focuses on this problem and reports the invariants and the properties extracted and verified using a series of tools (Daikon, MeCoSim, Maple, Spin, ProB) and languages (P–Lingua, Promela, Event-B). Appropriate tools and integration plugins, which facilitate and even automate the steps involved in the aforementioned approach, have also been developed. The case study chosen is complex (it involves an exponential growth of the number of states through the use of membrane division rules) and the properties obtained are non-trivial.

1 Introduction

Inspired by the behaviour and structure of the living cell, P systems have emerged in recent years as powerful computational tools [21]. Many variants of P systems have been introduced and a number of theoretical aspects have been intensely studied: the computational power of different variants, their capabilities to solve hard problems, like NP-complete ones, decidability, complexity aspects and hierarchies of classes of languages produced by these devices [23]. In the last years there have also been significant developments in using the P systems paradigm to model, simulate and formally verify various systems [5, 23].

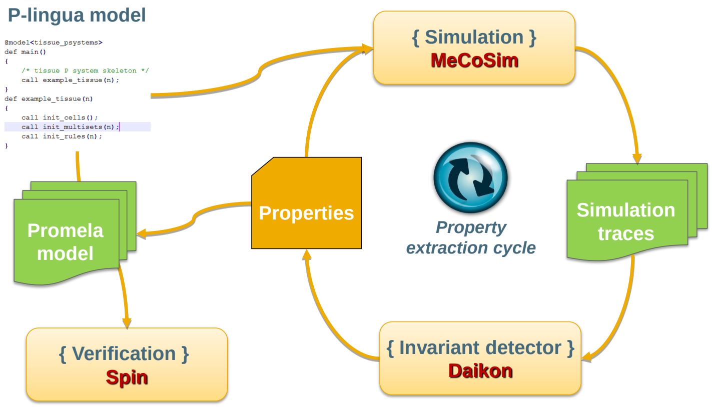
Up to now, two main areas concerning P system formal verification have been investigated: property verification through model checking and property extraction. However, there is no approach which integrates these two aspects (or other related aspects such as simulation). Initial research on P system model checking has tackled the problem of identifying decidable (or undecidable) problems [7, 8]. Verifying properties for P systems implies defining and implementing an operational semantics of the P system and using a corresponding model-checker. Among the tools used we mention: Maude [2], the probabilistic model-checker Prism [24, 4], the symbolic model verifier NuSMV [17], the Spin [19, 18] and ProB model checkers [16].

Property extraction using Daikon, a dynamic invariant detector, and further verification of the P systems was tackled in [4, 15]. In [4] a simple example involving a regulatory network is presented, along with the properties (preconditions, postconditions, invariants) inferred by Daikon. The relationships discovered regard the boundaries of the number of objects (e.g., $0 \leq prot \leq 205$) and relations between objects, such as $rna < orig(rep)$ or $(rna = 0) \rightarrow (prot = 0)$. They have been checked using the Prism probabilistic model checker. In [15] simple cell like P systems have been used and invariants like $2 * c - d = 0$, $(b = 0) \rightarrow (orig(b) = 0)$ have been obtained and further verified using NuSMV.

In this paper, we propose an integrated methodology for modelling, simulation, analysis, property extraction (invariant detection) and verification through model checking for P systems. The approach integrates a modelling and simulation environment (P–Lingua and MeCoSim) with model checkers, property extraction tools (Daikon) and tools for mathematical and symbolic calculus (Maple). Appropriate integration tools (plugins) have also been developed (see Fig. 1).

Starting from a problem, this process involves: the modelling of the problem by means of P systems, the model transcription into a language like P–Lingua [14], understandable by a machine; the definition of a visual interface, to enter the needed inputs and show the desired outputs from the computation; the simulation of the model under different initial parameters; the data extraction from the simulation; the invariants detection from the extracted data, and the analysis and verification of the detected properties. A detailed description of the methodology has been provided in section 4, applying the process until the invariants detection for the 3–Col problem in subsection 4.4. The approach is illustrated on a case study involving the tissue P system model for the well-known 3-colouring (3-Col) problem [10, 12].

The paper is structured as follows. We start by presenting in Section 2 the notation and main concepts to be used in the paper. Section 3 presents a set of initial properties for the 3-colouring problem, which have been verified using Spin and ProB model checker. In the next two sections are presented a methodology for properties extraction (invariants detection), its integration with the MeCoSim platform [20] and the empirical detection and validation of additional properties. Finally, conclusions are drawn in Section 6.



```
@model<tissue_psystems>
def main()
{
    /* tissue P system skeleton */
    call example_tissue(n);
}
def example_tissue(n)
{
    call init_cells();
    call init_multisets(n);
    call init_rules(n);
}
```

Fig. 1. Methodology Overview

2 Background

Before presenting our approach, let us establish the notations used and define the class of P systems addressed in the paper.

Given a finite alphabet $V = \{a_1, ..., a_p\}$, a *multiset* is either denoted by a string $u \in V^*$ (in which the order is not important, the string notation is only used as a convention), or by an associated vector of non-negative integers, $\Psi_V(u) = (|u|_{a_1}, ..., |u|_{a_p})$, where $|u|_{a_i}$ denotes the number of a_i occurrences in u, for each $1 \leq i \leq p$.

The following definition refers to a model of tissue P systems with cell division, introduced in [22]. This model can be seen as a network of cells, whose structure is not static: it is inspired by the way cells are duplicated in a natural way via mitosis.

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

$$\Pi = (\Gamma, w_1, \dots, w_q, \varepsilon, R, i_0),$$

where:

1. $q \geq 1$ *is the initial* degree *of the system; the system contains* q *cells, labelled with* $1, 2, \dots, m$*;* 0 *represents the environment.*

2. Γ is a finite alphabet (called working alphabet), whose symbols will be called objects.
3. $w_1, \dots, w_q$ are strings over Γ, describing the multisets of objects placed in the q cells of the system.
4. $\varepsilon \subseteq \Gamma$ is the set of objects present in the environment in arbitrarily many copies each.
5. R is a finite set of developmental rules *of the following form:*
   a) Communication rules: $(i, u/v, j)$, *for* $i, j \in \{0, 1, 2, \dots, q\}$, $i \neq j$, $u, v \in \Gamma^*$. *When applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and simultaneously the objects of the multiset v are sent from region j to region i.*
   b) Division rules: $[a]_i \rightarrow [b]_i[c]_i$, *where $i \in \{1, 2, \dots, q\}$ and $a, b, c \in \Gamma$. The cell with label i is divided in two cells with the same label; in the first copy the object a is replaced by b, in the second copy the object a is replaced by c; all other objects are replicated and copies of them are placed in the two new cells.*
6. $i_0 \in \{0, 1, 2, \dots, q\}$ *denotes the* output region *(which can be the region inside a membrane or the environment).*

Rules are applied as usual in a maximally parallel way, with 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 do not evolve in that step.

This class of P systems can be further extended, for solving NP-complete problems, to recognizer P systems. A *recognizer tissue P system with cell division* is a tuple $(\Gamma, \Sigma, w_1, \dots, w_q, \varepsilon, R, i_{in}, i_0)$, which has, in addition to a tissue P system with cell division:

- Two distinguished objects $yes, no \in \Gamma$, present in at least one copy in $w_1, w_2, \dots, w_q$, but not present in ε.
- An input alphabet Σ strictly contained in Γ.
- An input cell $i_{in} \in \{1, \dots, q\}$.

Also, it must satisfy the followings:

- The output region i_0 is the environment.
- All computations halt.
- If C is a computation of Π, then either the object *yes* or the object *no* (but not both) must have been released into the environment, and only in the last step of the computation.

3 Verifying a first set of properties for the 3-colouring problem

In this section we will introduce a simplified version of a tissue P system solving the 3-colouring problem, we will present some of its properties and their verification using the Spin and ProB model checkers.

3.1 A P system for the 3-colouring problem

In order to illustrate our approach regarding property extraction and verification for P systems, we have considered a simplified version of the 3-colouring problem from [9, 11].

The k-colouring problem is formulated as follows: given an undirected graph $G = (V, E)$, decide whether or not G is k-colourable; that is, if there exists a valid k-colouring of G (for every edge $\{u, v\} \in E$ the colours of u and v are different).

The 3-colouring problem can be solved in linear time by a family of recognizer tissue P systems with cell division [9]. The solution proposed in [9] is using a brute force algorithm, in the framework of recognizer tissue P systems with cell division, which consists of 4 stages:

1. Generation Stage: an initial cell, labelled by 2, is divided into two new cells; this process is repeated until all possible candidate solutions to the problem are generated (one solution for each membrane).
2. Prechecking Stage: after obtaining all possible 3-colourings (in cells labelled by 2), additional objects are generated in the cells, for every edge of the graph.
3. Checking Stage: it is verified if there exists a pair of adjacent vertices in the graph, with the same colour in the corresponding candidate solution.
4. Output Stage: the system sends to the environment the right answer according to the results of the previous stage (*yes* or *no*).

As we will focus in the rest of the paper only on the properties from the Generation Stage, we will omit from the recognizer P system model given in [9] some rules and objects, which would hinder the understanding of the mechanism. More precisely, we will consider only the division rules and a restricted set of objects, so we can define the model using the basic class of tissue P systems with cell division. However, for a complete specification in terms of a family of recognizer tissue P systems, [9] can be consulted.

Let $\Pi(n) = (\Gamma(n), w_1, w_2(n), \varepsilon, R(n), i_0)$ be a family of tissue P systems with cell division of degree 2, where:

1. $\Gamma(n) = \{A_i, R_i, T_i, B_i, G_i : 1 \leq i \leq n\}$
2. $w_1 = \emptyset$, $w_2(n) = \{A_1, \ldots, A_n\}$
3. $R(n)$ is a set of division rules:
   - $r_{1,i} \equiv [A_i]_2 \rightarrow [R_i]_2[T_i]_2$ for $i = 1, \ldots, n$
   - $r_{2,i} \equiv [T_i]_2 \rightarrow [B_i]_2[G_i]_2$ for $i = 1, \ldots, n$

In this model A_i encodes the i-th vertex of the graph; R_i, B_i, G_i represent the three colours red, blue, green. Appendix A presents two examples of computation for $\Pi(2)$ and $\Pi(3)$. It can be observed that, after appropriate divisions, in the step $2n$ we get exactly 3^n cells encoding all the possible 3-colourings of the graph having vertices $A_1, \ldots, A_n$. Appendix B presents the number of cells labelled 2 at each computation step for $2 \leq n \leq 11$, simulation results which help us formulate some interesting properties.

Given the family of P systems $\Pi(n)$ previously defined, an initial set of properties have been identified manually (without using property extraction tools):

P_1 For each computation C of the P system $\Pi(n)$, there are 3^n cells labelled with 2 at configuration C_{2n}.
P_2 For each computation C of the P system $\Pi(n)$, the configuration C_{n+1} has exactly $2^{n+1} - 1$ cells labelled 2.
P_3 For each computation C of the P system $\Pi(n)$, for each $0 \leq j \leq n$ the configuration C_j has exactly 2^j cells labelled 2.
P_4 For each combination $(X_1, X_2, \ldots, X_n)$, $X_i \in \{R_i, G_i, B_i\}$, $i = 1 \ldots n$, there exists, at configuration C_{2n}, one and only one cell labelled by 2 that contains the multiset $\{X_1, X_2, \ldots, X_n\}$.

3.2 3-Col property verification using model checking

This initial set of properties is now verified using two model checkers, Spin and ProB.

Spin is a model checker widely used in industries that build critical systems and is considered one of the most powerful model checkers available [3]. It is designed for modelling and verifying concurrent and distributed systems specified in Promela (Process or Protocol Meta Language), a verification modelling language. The properties to be verified can be expressed in LTL or by using assertion statements.

ProB is an animation and model checking tool integrated within the Rodin platform, which accepts Event-B models [1]. Unlike most model checking tools, ProB works on higher-level formalisms and so it enables a more convenient modelling. Besides verification of properties (expressed using the LTL or the CTL formalism), it also provides animation facilities, allowing to visualize, at any moment, the state space or to execute a given number of operations.

Property verification using Spin

As explained in [18], the executable specification for the Spin model checker associated to a P system will contain extra states and variables, corresponding to intermediate steps, which have no correspondence in the P system configurations. For this reason, the properties to be verified, that refer to the P system, need to be reformulated as equivalent LTL formulas for the associated Promela implementation.

The P system properties verified in this section are of the form 'G $(\phi \rightarrow \psi)$', and the equivalent LTL formula for the Promela model is '$[](!\phi \; || \psi || \; !pInS)$', as formally proven in [19] ($pInS$ is used to express if the current configuration in the Promela model represents also a state in the P system).

In our experiments, we have used one Promela specification file for each particular P system $\Pi(n)$, for all $n \in \{2, \ldots, 9\}$ because these files were (semi-)

automatically generated from each corresponding instance of P–Lingua definition file using the `plinguacore` library. We have successfully simulated all these models with Spin, for $n \in \{2, \ldots, 9\}$; typical state explosion problems appeared when we attempted to verify the properties mentioned earlier for $n \geq 4$.

For $n \in \{2, 3\}$ we have verified all the formulas presented bellow, using also some techniques that Spin provides to reduce the memory use. Starting with $n = 4$, we obtained `out of memory` for properties that involved checking many steps of the computations.

The properties verified, expressed as LTL formulas for the Spin model checker, for $n = 3$, are:

P_1 :[] $((!(noOfSteps == 6) \mid\mid (noOfCells == 27) \mid\mid (!pInS))$

P_2 :[] $((!(noOfSteps == 4) \mid\mid (noOfCells == 15) \mid\mid (!pInS))$

P_3 :[] $((!(noOfSteps >= 0 \;\&\&\; noOfSteps <= 3)) \mid\mid (noOfCells == pow2noOfSteps) \mid\mid (!pInS))$, where $pow2noOfSteps$ is a variable which computes $2^{noOfSteps}$

P_4 :This property is hard to verify with Spin because of the complex operations involved.

Property verification using ProB

The Event-B model of a P system with active membranes can be specified using two functions *cell* and *cellp*, representing the number of objects of each type contained in every cell and the number of objects produced between two steps of maximal parallelism, respectively. The rules are represented by events. Each division rule adds a cell to the domain of these functions. Additionally, a special event called *update*, enabled after each step of maximal parallelism, is used to add each value of *cellp* to *cell* and to reset all the values of *cellp* to 0. More details can be found in [6].

First, we developed an Event-B model, using the Rodin platform, for each particular P system $\Pi(n)$, with $n \in \{2, 3, 4\}$. Then, the possibility to use quantifiers in ProB allowed us to develop a general Event-B model for the family of P systems $\Pi(n)$, that has been instantiated for particular values of n. We animated the models in order to see how the system evolves and we verified their properties using the model checker ProB. Unlike the Promela specification, where the number of cells with label 2 was incremented after each division rule, we could specify the properties P_1-P_4 neglecting the extra intermediary) states; this is because these properties refer only to the number of steps of maximal parallelism in the evolution of $\Pi(n)$ (counted by a variable called *noOfSteps*) and to the number of cells with label two (counted by a variable called *noOfCells*), whose values are modified only in the *update* event. On the other hand, we used an additional state, *Halt*, to mark final configurations. However, we were able to verify all the properties only for $n \in \{2, 3, 4\}$; for $n = 5$, due to the state explosion problem, the model checker crashed with an out of memory error before reaching the final configuration (after producing 196 cells labelled 2). Consequently, for $n > 5$, we could verify only some simple properties, that do not involve terminal configuration (e.g. P_3 for small values of j).

The properties were specified using LTL as follows:

P_1 :$G\{state = Halt \Rightarrow noOfSteps = 2 * n \,\&noOfCells = 3^n\}$
P_2 :$G\{noOfSteps = n + 1 \Rightarrow noOfCells = 2^{n+1} - 1\}$
P_3 :$G\{!j.j >= 0 \,\&j <= n \,\&\, noOfSteps = j \Rightarrow noOfCells = 2^j\}$
P_4 :We were able to verify P_4 splitting it in two properties, the first one for the existence and the second one for the unicity. For $n = 2$ these properties were formulated as follows:

- For all x, y symbols in $\{R_i, G_i, B_i\}$, $i \in \{1, 2\}$, there exist one cell in the final configuration that contains one x and one y:
  $G\{state = Halt \Rightarrow (!x, y.x : \{R_1, R_2, G_1, G_2, B_1, B_2\} \,\&\, y : \{R_1, R_2, G_1, G_2, B_1, B_2\} \,\&\, (x/ = y) \Rightarrow (\#c.c : \mathit{dom}(\mathit{cell}) \,\&\, \mathit{cell}(c)(x) = 1 \,\&\, \mathit{cell}(c)(y) = 1))\}$
- In the final configuration any two different cells $c1$, $c2$ have different contents:
  $G\{state = Halt \Rightarrow (!c1, c2.c1 : \mathit{dom}(\mathit{cell}) \,\&\, c2 : \mathit{dom}(\mathit{cell}) \,\&\, (c1/ = c2) \Rightarrow (\#s.s : \{R_1, R_2, G_1, G_2, B_1, B_2\} \,\&\, \mathit{cell}(c1)(s)/ = \mathit{cell}(c2)(s)))\}$
  For higher values of n the formulas for these properties are very large and for space considerations, we will omit them.

Here "!", "#" and ":" correspond to the universal quantifier "∀", existential quantifier "∃" and membership operator "∈", respectively.

4 Integrating Daikon in MeCoSim and finding new relations

The next stage in the proposed methodology is the automatic extraction of new properties from simulation traces. The main tools used (MeCoSim and Daikon) and their integration are presented next. The process of identifying new properties, broken down in a number of individual steps, is also described.

4.1 Modelling and formalization

Once the 3-Col problem has been studied and modelled by means of P systems, this model must be expressed in a language that may be understood by a simulation machine. For this purpose, the standard P–Lingua [14] language has been chosen as our modelling language. The initial multisets and rules are expressed as follows:

```
@ms(2) += A{i} : 1<=i<=n;

/* r1 */ [A{i}]'2 --> [R{i}]'2 [T{i}]'2 : 1<=i<=n;
/* r2 */ [T{i}]'2 --> [B{i}]'2 [G{i}]'2 : 1<=i<=n;
```

The full code of the model in P–Lingua is showed in Appendix C. This file will serve as an input for MeCoSim, so that the P system can be simulated, analyzed and debugged, and invariants can be detected, to be then verified by model checking.

4.2 MeCoSim

In order to provide an integrated methodology for model simulation, properties extraction and verification, we need an integrated environment to simplify the user's process.

In this sense, a general purpose membrane computing simulator, **MeCoSim** [20], was provided. It was initially designed to enable the user defined customized interfaces, with inputs, outputs, charts, etc., adapted to each family of P systems. This permits entering data for different initial conditions, instantiating different P systems of the family.

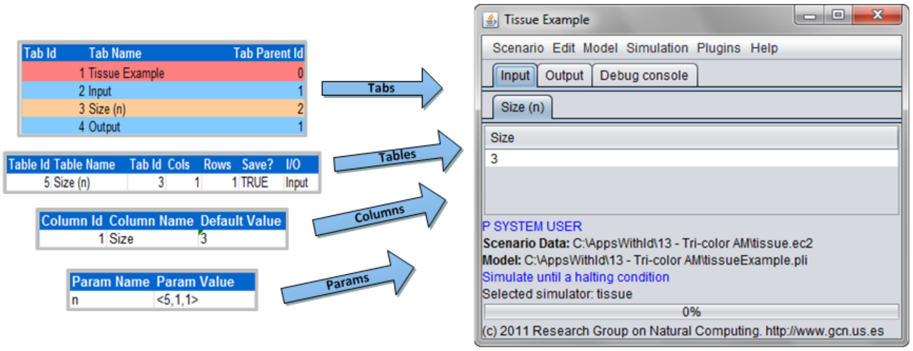
The initial aim of this software environment has been extended such that it can cover a more general set of applications by providing flexible and powerful methods to integrate various software applications and packages as MeCoSim plugins. These kind of plugins can be easily added to MeCoSim by setting appropriate parameters in a configuration file. Keeping in mind this architecture and the developed plugins, MeCoSim may provide a platform for the integration of different tools for the modelling, simulation, analysis, property extraction and verification of P systems. Some of this tools have already been developed and/or integrated, others are being developed, and many other could be added in a similar way.

To take advantage of this framework for studying the 3-Col problem, we need to define our customized inputs, outputs, extractions, etc. The main steps of this process are illustrated in the next paragraphs.

MeCoSim permits setting the hierarchy of tabs to be shown in the visual user interface, including input and output tables inside each leaf tab. In our case, we divide the information in two tabs, Input and Output (plus an additional tab, Debug console, provided by default in MeCoSim, used for debugging the models). For our example, we only need one input parameter, n, so one input table is defined inside the tab Input, as showed in Fig. 2.

Now the simulation could be performed, in such a way that n takes the value from the input table, this parameter complements the P–Lingua file to instantiate the initial configuration of the P system and the computation steps run until a halting condition is reached. In the debug console, we can run step by step, looking at all the objects of the multisets inside each cell and compartment, for each computation step.

However, this process could be very slow if we are interested in several objects, membranes or steps, so we need a way to define customized outputs showing the desired information only. MeCoSim provides this mechanism, and we define outputs as shown in Fig. 3 to study different issues: entire configurations, objects per membrane, objects by type (R, G, B), number of cells, etc.


Tabs
Tables
Columns
Params
Tissue Example
Scenario Edit Model Simulation Plugins Help
Input Output Debug console
Size (n)

Tab Id	Tab Name	Tab Parent Id
1	Tissue Example	0
2	Input	1
3	Size (n)	2
4	Output	1

Table Id	Table Name	Tab Id	Cols	Rows	Save?	I/O
5	Size (n)	3	1	1	TRUE	Input

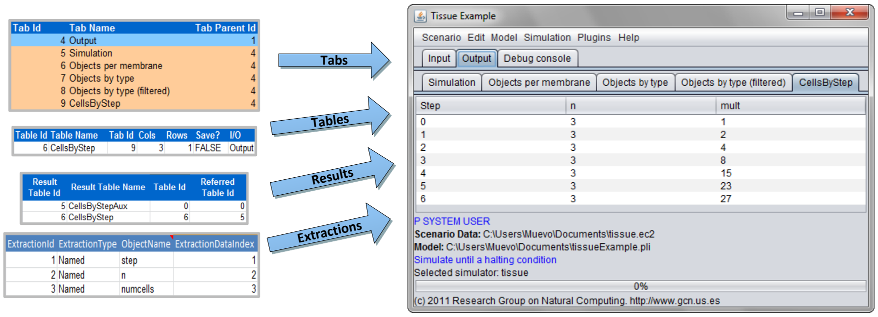
Column Id	Column Name	Default Value
1	Size	3

Param Name	Param Value
n	<5,1,1>

Size
3

P SYSTEM USER
Scenario Data: C:\AppsWithId\13 - Tri-color AM\tissue.ec2
Model: C:\AppsWithId\13 - Tri-color AM\tissueExample.pli
Simulate until a halting condition
Selected simulator: tissue
0%
(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Fig. 2. MeCoSim window - Input tab: value of n


Tabs
Tables
Results
Extractions
Tissue Example
Scenario Edit Model Simulation Plugins Help
Input Output Debug console
Simulation Objects per membrane Objects by type Objects by type (filtered) CellsByStep

Tab Id	Tab Name	Tab Parent Id
4	Output	1
5	Simulation	4
6	Objects per membrane	4
7	Objects by type	4
8	Objects by type (filtered)	4
9	CellsByStep	4

Table Id	Table Name	Tab Id	Cols	Rows	Save?	I/O
6	CellsByStep	9	3	1	FALSE	Output

Result Table Id	Result Table Name	Table Id	Referred Table Id
5	CellsByStepAux	0	0
6	CellsByStep	6	5

ExtractionId	ExtractionType	ObjectName	ExtractionDataIndex
1	Named	step	1
2	Named	n	2
3	Named	numcells	3

Step	n	mult
0	3	1
1	3	2
2	3	4
3	3	8
4	3	15
5	3	23
6	3	27

P SYSTEM USER
Scenario Data: C:\Users\Muevo\Documents\tissue.ec2
Model: C:\Users\Muevo\Documents\tissueExample.pli
Simulate until a halting condition
Selected simulator: tissue
0%
(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Fig. 3. MeCoSim window - Output tab: number of cells by step

Eventually, as we see in Fig. 3, we set the information to be extracted for Daikon. This makes up the output files from the simulation, to be used as an input for the MeCoSim Daikon plugin.

4.3 MeCoSim new plugin for Daikon integration

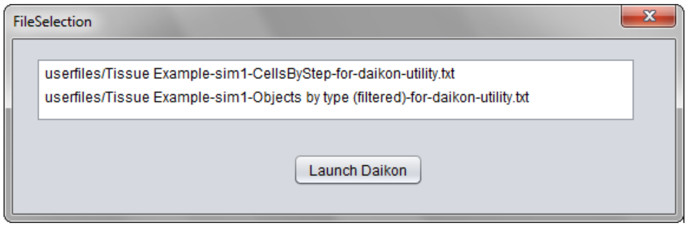
MeCoSim provides an easy way to add plugins, enriching the default functionality. Taking advantage of this architecture, a new plugin has been developed to integrate Daikon with MeCoSim.

Daikon [13] is a tool which dynamically detects programs invariants, based on their execution traces. It can discover properties from C, C++, Eiffel, Java, or Perl programs, from spreadsheet files and other data sources. The usual operation of Daikon is the following: it receives data trace files about the values of some variables across a sequence of steps from the execution of a program, and tries to detect properties of types: precondition, postcondition and invariant. We are mainly interested in the last one, but the previous ones could be also useful for checking the correctness of the models.

As part of the proposed methodology, we aim to integrate this tool with MeCoSim, so invariants could be detected from the desired outputs of the simulation. For this purpose, a plugin has been developed; an overview of the entire (simulation and property extraction) process involves the following steps:

1. The model of the P system (written in P–Lingua, possibly parametrized) is loaded in MeCoSim.
2. The initial parameters (n in the case of the 3–Col problem) for instantiating the specific P system are provided by the user in a visual way through the input tables.
3. The simulation runs, generating extraction files for the outputs previously set.
4. The plugin can be called from a menu option (Plugins > "Daikon").

When the plugin is launched from MeCoSim, a window with a listing of available extraction files is visualized, as showed in figure 4.



userfiles/Tissue Example-sim1-CellsByStep-for-daikon-utility.txt
userfiles/Tissue Example-sim1-Objects by type (filtered)-for-daikon-utility.txt

Fig. 4. MeCoSim window - Daikon plugin - File selection

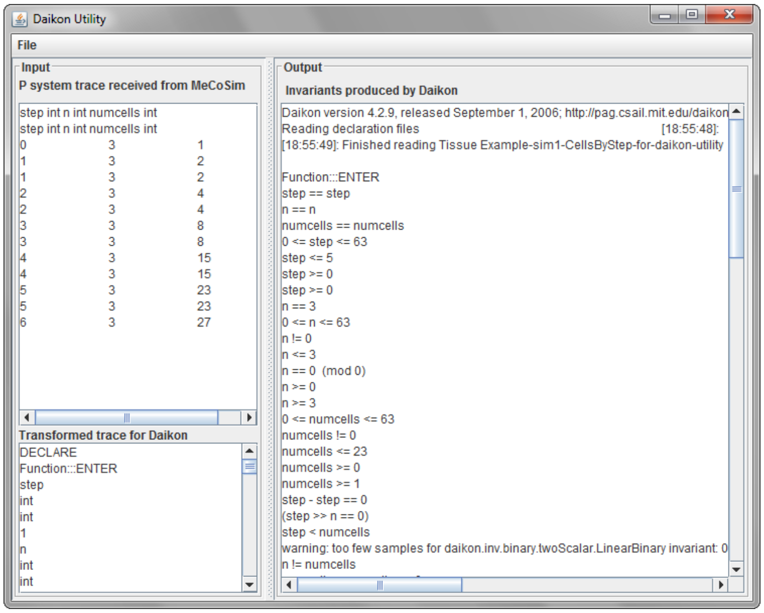
Once one of the extraction files is selected, the Daikon plugin runs from this input file. It automatically reads the simulation extraction file, generates the traces in the appropriate format and launches Daikon from this traces file, trying to detect as many invariants as possible. They are eventually visualized, as showed in figure 5.

Further technical details concerning the Daikon plugin and its integration into MeCoSim is provided in Appendix D.

4.4 Methodology

In the previous sections, different tools and languages for modelling, simulation, verification and invariant detection have been explained. In order to integrate the different tools in a systematic way, a novel methodology has been devised, as outlined in Fig. 1.

In the first stage, we need to model the problem, writing the model in P–Lingua, Promela and other languages, in order to serve as the default input for Spin and other possible model checkers.


Daikon Utility
File
Input
P system trace received from MeCoSim
Output
Invariants produced by Daikon
Transformed trace for Daikon

```
step int n int numcells int
step int n int numcells int
0              3              1
1              3              2
1              3              2
2              3              4
2              3              4
3              3              8
3              3              8
4              3              15
4              3              15
5              3              23
5              3              23
6              3              27
```

```
DECLARE
Function:::ENTER
step
int
int
1
n
int
int
```

```
Daikon version 4.2.9, released September 1, 2006; http://pag.csail.mit.edu/daikon
Reading declaration files                                        [18:55:48]:
[18:55:49]: Finished reading Tissue Example-sim1-CellsByStep-for-daikon-utility

Function:::ENTER
step == step
n == n
numcells == numcells
0 <= step <= 63
step <= 5
step >= 0
step >= 0
n == 3
0 <= n <= 63
n != 0
n <= 3
n == 0  (mod 0)
n >= 0
n >= 3
0 <= numcells <= 63
numcells != 0
numcells <= 23
numcells >= 0
numcells >= 1
step - step == 0
(step >> n == 0)
step < numcells
warning: too few samples for daikon.inv.binary.twoScalar.LinearBinary invariant: 0
n != numcells
```

Fig. 5. Daikon plugin - Invariants detection

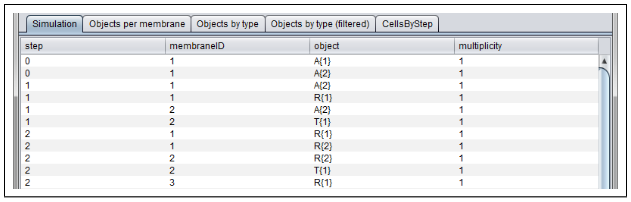
Once we have modelled the problem, written in P–Lingua, set the customized MeCoSim-based application including Daikon and entered the P–Lingua file in MeCoSim, the property extraction cycle can start.

For each iteration, some interesting goals should be addressed, for example to obtain relations between P system objects or invariants regarding the number of cells at each computation step. Then the needed outputs have to be set in MeCoSim, according to the stated objective. Based on the model and the target data entered in the input tables, simulations can be performed, obtaining the required results, which can be displayed and exported into the extraction files. The Daikon plugin can be executed further, in order to detect invariants from the extraction files, which contain traces from the P system simulations. Eventually, the new invariants detected by Daikon should be tested with the model checkers, to be verified.

In the following paragraphs this methodology is illustrated with some iterations for the 3–Col problem.

Iteration 1: property extraction using the entire model

The **goal** in this iteration is to analyze the values of all the objects across the simulation. A first output is set in MeCoSim (see Fig. 6), along with an extraction file for Daikon. No interesting properties (invariants) were found, so other alternative studies were considered.

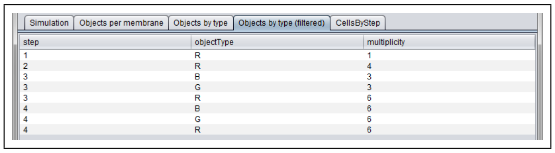


step	membraneID	object	multiplicity
0	1	A{1}	1
0	1	A{2}	1
1	1	A{2}	1
1	1	R{1}	1
1	2	A{2}	1
1	2	T{1}	1
2	1	R{1}	1
2	1	R{2}	1
2	2	R{2}	1
2	2	T{1}	1
2	3	R{1}	1

Fig. 6. Iteration 1. Output: entire simulation

Iteration 2: property extraction using the simplified model

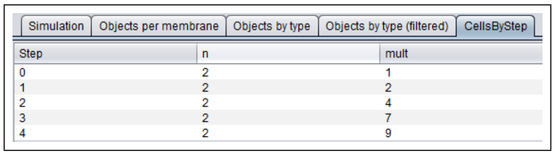
The **goal** in this iteration is to simplify the model and to filter the information sent to Daikon, for example by grouping the objects by type, or restricting the kind of objects analysed to R, G or B (and not considering the others, such as A and T). Again, the output is set in MeCoSim (see Fig. 7), along with an extraction for Daikon, and no interesting properties were found.



step	objectType	multiplicity
1	R	1
2	R	4
3	B	3
3	G	3
3	R	6
4	B	6
4	G	6
4	R	6

Fig. 7. Iteration 2. Output: objects by type (R, G, B)

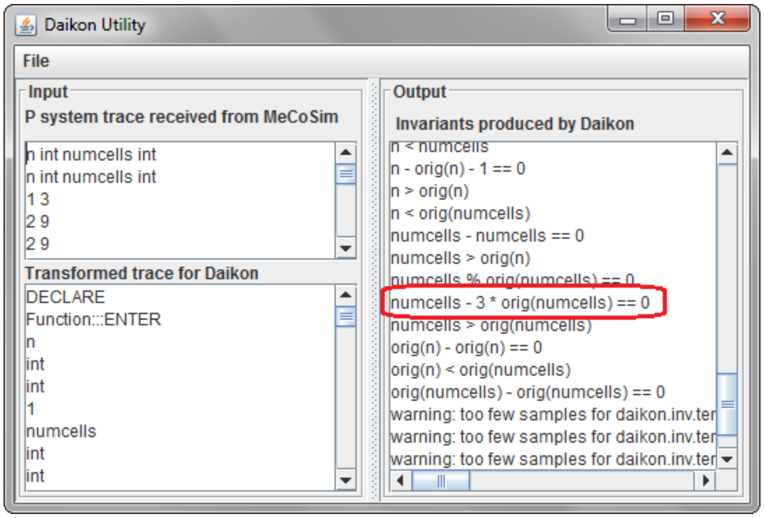
Another **goal** of this iteration is to obtain information (general formulas) regarding total number of cells in the P system for each computation step. To accomplish this, the respective output (see Fig. 8) and extraction are set. The simulation runs for $n = 2$, generating the extraction file, and Daikon Plugin is executed, but no relevant properties are obtained. The process is repeated for input $n = 3, 4, 5, \ldots$, but again no interesting properties are found.



Step	n	mult
0	2	1
1	2	2
2	2	4
3	2	7
4	2	9

Fig. 8. Iteration 2. Output: Cells by step

Instead of continuing with simulations for other values of n, it emerges the **idea** of collecting the results for different values of n in the same file, only for the last computation step, in order to get a general property, met for all the cases. The result is showed in figure 9. As it can be seen, an interesting invariant is detected: in the last configuration, having n as the input, $numcells = 3^n$ (Daikon indicates the equivalent recursive relation, $numcells - 3 * orig(numcells) == 0$).



P system trace received from MeCoSim

```
n int numcells int
n int numcells int
1 3
2 9
2 9
```

Transformed trace for Daikon

```
DECLARE
Function:::ENTER
n
int
int
1
numcells
int
int
```

Invariants produced by Daikon

```
n < numcells
n - orig(n) - 1 == 0
n > orig(n)
n < orig(numcells)
numcells - numcells == 0
numcells > orig(n)
numcells % orig(numcells) == 0
numcells - 3 * orig(numcells) == 0
numcells > orig(numcells)
orig(n) - orig(n) == 0
orig(n) < orig(numcells)
orig(numcells) - orig(numcells) == 0
warning: too few samples for daikon.inv.ter
warning: too few samples for daikon.inv.ter
warning: too few samples for daikon.inv.ter
```

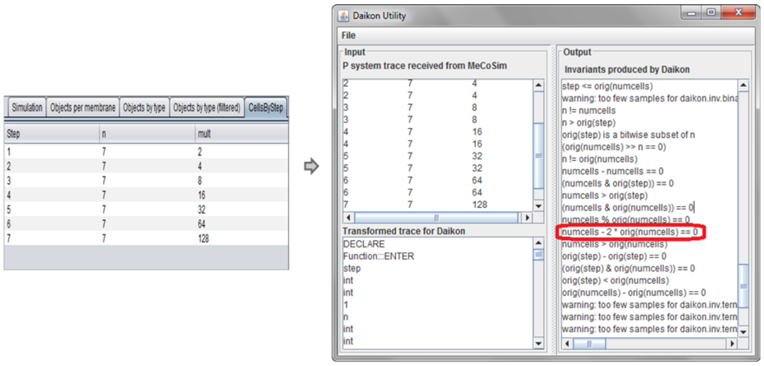
Fig. 9. Iteration 2. Invariants detection: Cells in last configuration

Iteration 3: dividing the computation path and extracting properties from the sub-paths, according to the model features

Another **idea** for extracting relevant information regarding the P system is to analyse only the configurations from a certain computation sub-path. In the case of the 3-Col problem, we anticipate that for the first half of the computation, the number of cells at each step is a power of 2.

The first **goal** in this iteration is to count the number of cells for each step until half the computation. The simulation runs, for example, for $n = 7$, generating the extraction file, and Daikon Plugin is executed, getting the results showed in Fig. 10. A new important invariant has been detected: for each $step$ until half the computation (that is, until $step = n$), $numcells = 2^{step}$ (Daikon indicates the equivalent recursive relation, $numcells - 2 * orig(numcells) == 0$).

The second **goal** in this iteration is to count the number of cells for each step in the second half of the computation. The output and extraction are set, the simulation runs again for $n = 7$, generating the extraction file. Daikon Plugin is executed, detecting the invariant: for each $step$ from half the computation to the end , $numcells = 3 (mod\, 4)$. However, this invariant was verified with the model checkers, resulting that this is not true for all the values of n, so it was not validated and cannot be considered a general property.



Step	n	mult
1	7	2
2	7	4
3	7	8
4	7	16
5	7	32
6	7	64
7	7	128

P system trace received from MeCoSim

```
2      7      4
2      7      4
3      7      8
3      7      8
4      7      16
4      7      16
5      7      32
5      7      32
6      7      64
6      7      64
7      7      128
```

Transformed trace for Daikon

```
DECLARE
Function:::ENTER
step
int
int
1
n
int
int
```

Invariants produced by Daikon

```
step <= orig(numcells)
warning: too few samples for daikon.inv.bina
n != numcells
n > orig(step)
orig(step) is a bitwise subset of n
(orig(numcells) >> n == 0)
n != orig(numcells)
numcells - numcells == 0
(numcells & orig(step)) == 0
numcells > orig(step)
(numcells & orig(numcells)) == 0
numcells % orig(numcells) == 0
numcells - 2 * orig(numcells) == 0
numcells > orig(numcells)
orig(step) - orig(step) == 0
(orig(step) & orig(numcells)) == 0
orig(step) < orig(numcells)
orig(numcells) - orig(numcells) == 0
warning: too few samples for daikon.inv.tern
warning: too few samples for daikon.inv.tern
warning: too few samples for daikon.inv.tern
```

Fig. 10. Iteration 3. Invariants detection: Cells each last step, first half of the computation

As it has been seen, the methodology includes some important parts that have been integrated, automating the process of modelling, simulating, analyzing, debugging and detecting invariants from the MeCoSim application, making use of the simulation engine of pLinguaCore and the invariants detector Daikon. However, some parts of the process are being used independently, and have not been integrated with MeCoSim until the moment. The methodology covers all the process, and we plan to add the needed plugins to let it be as automated as possible.

4.5 Results - Summary of discovered properties

The idea of using simulation traces to infer properties of the P system model, as detected by Daikon, is useful in order to check the correct behaviour of the system and in the same time to find out new relationships between model variables. We can classify the results proposed by Daikon into:

- *Obvious invariants:* these confirm that the model is behaving as it should. For example, some results obtained for $n = 10$ are:
  - $B >= 0$, $B <= 196830$: the number of objects will never be negative, even more the sum of B_i objects over all membranes is at maximum 196830. The last relation is correct, moreover we estimate that the total number of occurrences of objects of type R_i, G_i, B_i at the end of computation is $n \times 3^{n-1}$ and $196830 = 10 \cdot 3^9$. This property has been verified with Spin for $n \in \{2, 3\}$ and with ProB for $n \in \{2, 3, 4\}$. For higher values of n, as stated in section 3.2 this property could not be verified because it involves checking many steps of the computations which yields to the “out of memory” problem.
  - $(step\ == 0) ==> (B\ == 0)$ is obvious because the initial multiset is $w_2 = A_1 \dots A_n$.

  - $B <= R$ is a direct consequence of the fact that rules $[A_i]_2 \to [R_i]_2[T_i]_2$ are applied first, so R_i objects are produced, and later are applied rules of type $[T_i]_2 \to [B_i]_2[G_i]_2$
  - $B >= orig(B)$ is obvious because B_i objects are never consumed by any rules.
- *Anomalous invariants:* these indicate a fault in the model and its parameter values. In this case, we did not obtained any of these, but if the P–Lingua model would have been incorrect (not solving the proposed problem), we could have encountered anomalous values.
- *Interesting invariants:* could even suggest novel relationships between the model variables. However, these properties should be further confirmed by a model checker verification.

A summary of the extracted properties is presented in Table 1, where by non-interesting properties we refer to obvious invariants, such the ones presented previously. Also, the truth value given in the third column refers to the result returned by the model checkers, after verifying the corresponding properties.

Extraction	**Result**	**Truth**
Entire model (all simulation data)	Non-interesting properties	
Objects grouped by type	Non-interesting properties	
Objects grouped by type, filtered by R, G or B	Non-interesting properties	
No. of cells for each n separately	Non-interesting properties	
No. of cells in the last configuration, for different values of n together	$numcells = 3^n$	true
No. of cells for each step $0 \ldots n$	$numcells = 2^{step}$	true
No. of cells for each step $(n+1) \ldots 2n$	$numcells = 3(mod\, 4)$	false
No. of cells for each step $(n+1) \ldots (n+(n/2)+1)$	$numcells = 3(mod\, 12)$	false

Table 1. Summary of detected invariants

5 Discovering other properties with Maple

5.1 Model simplification for simulation

As we have discussed in Section 3.2, verification through model checking is possible for n up to 5. Beyond this point, due to the state explosion problem, both model checkers (Spin and ProB) will crash with an out of memory error, at least for some of the verified properties. This led us to build and use a simplified model of the investigated P system, based on the observation that the evolution of the number of cells and the actual contents of each cell are two separable issues, because only the symbols A and T appear in the left hand side of the rules. Thus, the number of objects R, G, B was ignored; first, a simplified Event-B model was produced, but with only limited performance gains, i.e. verification of properties for n up to 6; secondly, a Python implementation was developed, which enabled the verification of properties for n up to 19. Based on the backtracking technique, the simulation Algorithm 1 follows the evolution of a cell placed in the top of a stack, across the configurations of $\Pi(n)$ for as long as it contains non-terminals. Each cell is stored as a tuple containing the multiplicity of A's and T's and the step number. If the current cell still contains terminals, a new cell is added on to the stack, otherwise it is removed. We also used an array to count the number of cells produced at each step. The algorithm ends when the stack is empty. The verification Algorithm 2 checks if the values from the array calculated in the first algorithm are the same with the values returned by a function implemented for each property. Both algorithms are given in Appendix E.

5.2 Obtaining the polynomial coefficients with Maple

As we have seen in Section 4.4, using Daikon we have managed to find a number of (simpler) invariants but have failed to find other potentially interesting properties, such as the number of cells from configurations $n + 2$ up to configuration $2 * n$. This is due to the quite complex nature of the formulae for these numbers: Using the number of cells with label 2 given in Appendix B, we deduced that the number of cells in configuration $n + k$, $k \in \{2, 3, 4, 5, \ldots\}$, is a sum between 2^{n+k} and a polynomial $Q_k$of degree $k - 1$. In order to determine the exact expression of this polynomial we used Maple.

Maple is a powerful software that can be used to solve various mathematical problems with numerical and symbolic calculus. It also incorporates a programming language that allows working with formulas containing symbols and formal operations. Maple provides users over 5000 predefined functions and commands, with suggestive names, dedicated to various branches of mathematics.

In order to obtained Q_k we used the idea that a polynomial of degree $k - 1$ can be obtained solving a recurrence of order k and some Maple functions and commands:

- $rgf\_findrecur(k, seq, f, n)$: function contained in the package *genfunc* that finds the linear recurrence with constant coefficients of order k that is satisfied by the sequence with $2k$ terms *seq*; f is the name of the general term and n is the index variable of the recurrence;
- $rsolve(\{rec\}, f(n))$: command returning an expression for the general term of the function $f(n)$ by solving the recurrence *rec*;

- *expand*(*expr*): command used to distribute products over sums in the given expression.

For $k = 4$, using $rgf\_findrecur(4, [(81 - 2^8), (227 - 2^9), (585 - 2^{10}), (1403 - 2^{11}), (3185 - 2^{12}), (6947 - 2^{13}), (14729 - 2^{14}), (30619 - 2^{15})], f, t)$ we obtained the recurrence: $f(t) = 4 * f(t-1) - 6 * f(t-2) + 4 * f(t-3) - f(t-4)$. Solving this recurrence with the command $rsolve(\{f(t) = 4*f(t-1)-6*f(t-2)+4*f(t-3)-f(t-4), f(4) = -175, f(5) = -285, f(6) = -439, f(7) = -645\}, f(n))$ we obtain the general term: $-19+12(n+1)\left(\frac{1}{2}n+1\right)-14n-8(n+1)\left(\frac{1}{2}n+1\right)\left(\frac{1}{3}n+1\right)$, and expanding it: $-15 - \frac{32}{3}n - 2n^2 - \frac{4}{3}n^3$. Adding 2^{n+4} to this general term we obtain the number of cells with label 2 in the configuration $n+4$. The recurrences from Table 2 have been obtained similarly.

Configuration	Recurrence
$n+2$	$f(t) = 2 \cdot f(t-1) - f(t-2)$
$n+3$	$f(t) = 3 \cdot f(t-1) - 3 \cdot f(t-2) + f(t-3)$
$n+4$	$f(t) = 4 \cdot f(t-1) - 6 \cdot f(t-2) + 4 \cdot f(t-3) - f(t-4)$
$n+5$	$f(t) = 5 \cdot f(t-1) - 10 \cdot f(t-2) + 10 \cdot f(t-3) - 5 \cdot f(t-4) + f(t-5)$
$n+6$	$f(t) = 6 \cdot f(t-1) - 15 \cdot f(t-2) + 20 \cdot f(t-3)$ $-15 \cdot f(t-4) + 6 \cdot f(t-5) - f(t-6)$

Table 2. Recurrence for the number of cells with label 2 in configuration $n + k$

We notice that all these recurrences have, for each configuration $n + k$, the following form: $f(t) = \sum_{i=1}^{k}(-1)^{i+1} \cdot C_k^i \cdot f(t-i)$. Unfortunately, the current limitations of our tools ($n < 20$) does not allow us to continue the calculus with the next steps and so we cannot establish unequivocally if the recurrences have the same form for larger values of k.

Solving these recurrences (using *rsolve*), expanding the expressions (with *expand*) and adding 2^{n+k}, we obtain the number of cells with label 2 in the configuration $n + k$, for $k \in \{2, 3, 4, 5, 6\}$, as presented in Table 3.

All these formulas were verified using Algorithm 2 from Appendix E for $n < 20$. As we can see from Table 3 the coefficient of the polynomial that follows 2^{n+k} are rational but we could not establish any further rule for them except the fact that, in each case, the free term is $-2^k + 1$.

Configuration	Number of cells
$n+2$	$2^{n+2} - 2n - 3$
$n+3$	$2^{n+3} - 2n^2 - 4n - 7$
$n+4$	$2^{n+4} - \frac{4}{3} \cdot n^3 - 2 \cdot n^2 - \frac{32}{3} \cdot n - 15$
$n+5$	$2^{n+5} - \frac{2}{3} \cdot n^4 - \frac{28}{3} \cdot n^2 - 20 \cdot n - 31$
$n+6$	$2^{n+6} - \frac{4}{15} \cdot n^5 + \frac{2}{3} \cdot n^4 - \frac{20}{3} \cdot n^3 - \frac{32}{3} \cdot n^2 - \frac{676}{15} \cdot n - 63$

Table 3. Number of cells with label 2 in configuration $n+k$

6 Conclusion

In this paper, we have outlined an integrated methodology for P system formal verification, comprising modelling using P–Lingua, simulation with MeCoSim, properly extraction using Daikon and model checking using tools such as Spin and ProB. A plugin which allows Daikon to be called and used within MeCoSim has been developed and a (semi)-automatic Promela implementation has been generated from the P–Lingua model. A number of steps involved in property extraction using Daikon have been identified and the whole process has been illustrated with an example, a tissue P system model of the 3-colouring problem; this is a complex problem since, by using active membranes (cell division), the number of cells grows exponentially. As some of the sought properties have proved to be quite complex and could not be directly extracted using Daikon, a tool for mathematical and symbolic calculus (Maple) has been used to supplement our methodology.

Further work involves the development of completely integrated environment for automatic modelling, simulation and verification of P systems as well as applying the proposed methodology to other, more complex, P systems.

Acknowledgment

The work of MG, FI and RL was supported by a grant of the Romanian National Authority for Scientific Research, CNCS–UEFISCDI, project number PN-II-ID-PCE-2011-3-0688. The authors MPJ and LVC acknowledge the support of the project TIN2009–13192 of the Ministerio de Ciencia e Innovación of Spain, co-financed by FEDER funds, and the support of the Project of Excellence with Investigador de Reconocida Valía of the Junta de Andalucía, grant P08–TIC–04200. AT was supported by project POSDRU - "Dezvoltarea şcolilor doctorale prin acordarea de burse tinerilor doctoranzi cu frecvenţă"- 88/1.5/S/52826.

References

1. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
2. Andrei, O., Ciobanu, G., Lucanu, D.: Executable specifications of P systems. In: Mauri, G., Păun, G., Pérez-Jiménez, M., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 3365, pp. 126–145. Springer Berlin / Heidelberg (2005)
3. Ben-Ari, M.: Principles of the Spin Model Checker. Springer-Verlag, London (2008)
4. Bernardini, F., Gheorghe, M., Romero-Campero, F.J., Walkinshaw, N.: A hybrid approach to modeling biological systems. In: Eleftherakis, G., Kefalas, P., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 138–159. Springer (2007)
5. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): Applications of Membrane Computing. Natural Computing Series, Springer (2006)
6. Ţurcanu, A., Ipate, F.: Modelling, testing and verification of P systems with active membranes using Rodin and ProB modelling. In: Proceedings of the 12th International Conference on Membrane Computing. pp. 459–468. Paris-Est University Press (2011)
7. Dang, Z., Ibarra, O., Li, C., Xie, G.: On model-checking of P systems. In: Calude, C., Dinneen, M., Păun, G., Pérez-Jímenez, M., Rozenberg, G. (eds.) Unconventional Computation, Lecture Notes in Computer Science, vol. 3699, pp. 82–93. Springer Berlin / Heidelberg (2005)
8. Dang, Z., Ibarra, O.H., Li, C., Xie, G.: On the decidability of model-checking for P systems. Journal of Automata, Languages and Combinatorics 11(3), 279–298 (2006)
9. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A linear-time tissue P system based solution for the 3-coloring problem. Electr. Notes Theor. Comput. Sci. 171(2), 81–93 (2007)
10. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A logarithmic bound for solving Subset Sum with P systems. In: Eleftherakis, G., Kefalas, P., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 257–270. Springer (2007)
11. Díaz-Pernil, D., Gutiérrez-Naranjo, 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. Theor. Comput. Sci. 404(1-2), 76–87 (2008)
12. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5391, pp. 187–203. Springer (2009)
13. Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically discovering likely program invariants to support program evolution. IEEE Trans. Software Eng. 27(2), 99–123 (2001)
14. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An overview of P-Lingua 2.0. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 10th International Workshop, WMC 2009, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5957, pp. 264–288. Springer (2010)

15. Gheorghe, M., Ipate, F., Lefticaru, R., Dragomir, C.: An integrated approach to P systems formal verification. In: CMC11: Proceedings of the Eleventh International Conference on Membrane Computing. pp. 225–238. ProBusiness Verlag Berlin (2010)
16. Ipate, F., Ţurcanu, A.: Modelling, verification and testing of P systems using Rodin and ProB. In: Ninth Brainstorming Week on Membrane Computing (BWMC 2011). pp. 209–220 (2011)
17. Ipate, F., Gheorghe, M., Lefticaru, R.: Test generation from P systems using model checking. Journal of Logic and Algebraic Programming 79(6), 350–362 (2010)
18. Ipate, F., Lefticaru, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Tudose, C.: Formal verification of p systems with active membranes through model checking. In: Gheorghe, M., Paun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) Int. Conf. on Membrane Computing. Lecture Notes in Computer Science, vol. 7184, pp. 215–225. Springer (2011)
19. Ipate, F., Lefticaru, R., Tudose, C.: Formal verification of P systems using Spin. International Journal of Foundations of Computer Science 22(1), 133–142 (2011)
20. Pérez-Hurtado, I., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Colomer, M.A., Riscos-Núñez, A.: Mecosim: A general purpose software tool for simulating biological phenomena by means of p systems. IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010) I, 637–643 (2010), `http://www.cs.us.es/~marper/investigacion/mecosim.pdf`
21. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
22. Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Tissue P system with cell division. In: Păun, G., Riscos-Núñez, A., Romero-Jiménez, A., Sancho-Caparrini, F. (eds.) Second Brainstorming Week on Membrane Computing. vol. Report RGNC 01/2004, pp. 308–386 (2004)
23. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
24. Romero-Campero, F.J., Gheorghe, M., Bianco, L., Pescini, D., Pérez-Jiménez, M.J., Ceterchi, R.: Towards probabilistic model checking on P systems using PRISM. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 7th International Workshop, WMC 2006, Revised, Selected, and Invited Papers. Lecture Notes in Computer Science, vol. 4361, pp. 477–495. Springer (2006)

Appendices

A Computation examples for $\Pi(n)$, $n \in \{2, 3\}$

Crt. step	**No. of cells labelled 2**	**Current configuration (only cells labelled with 2)**
0	1	$\left([A_1A_2]_2 \right)$
1	2	$\left([R_1A_2]_2\ [T_1A_2]_2 \right)$
2	4	$\left([R_1R_2]_2\ [R_1T_2]_2\ [B_1A_2]_2\ [G_1A_2]_2 \right)$
3	7	$\begin{pmatrix} [R_1R_2]_2\ [R_1B_2]_2\ [R_1G_2]_2\ [B_1R_2]_2\ [B_1T_2]_2 \\ [G_1R_2]_2\ [G_1T_2]_2 \end{pmatrix}$
4	9	$\begin{pmatrix} [R_1R_2]_2\ [R_1B_2]_2\ [R_1G_2]_2\ [B_1R_2]_2\ [B_1B_2]_2 \\ [B_1G_2]_2\ [G_1R_2]_2\ [G_1B_2]_2\ [G_1G_2]_2 \end{pmatrix}$

Table 4. Computation example for $\Pi(2)$

Crt. step	**No. of cells labelled 2**	**Current configuration (only cells labelled with 2)**
0	1	$\left([A_1A_2A_3]_2 \right)$
1	2	$\left([R_1A_2A_3]_2\ [T_1A_2A_3]_2 \right)$
2	4	$\left([R_1R_2A_3]_2\ [R_1T_2A_3]_2\ [B_1A_2A_3]_2\ [G_1A_2A_3]_2 \right)$
3	8	$\begin{pmatrix} [R_1R_2R_3]_2\ [R_1R_2T_3]_2\ [R_1B_2A_3]_2\ [R_1G_2A_3]_2 \\ [B_1R_2A_3]_2\ [B_1T_2A_3]_2\ [G_1R_2A_3]_2\ [G_1T_2A_3]_2 \end{pmatrix}$
4	15	$\begin{pmatrix} [R_1R_2R_3]_2\ [R_1R_2B_3]_2\ [R_1R_2G_3]_2\ [R_1B_2R_3]_2\ [R_1B_2T_3]_2 \\ [R_1G_2R_3]_2\ [R_1G_2T_3]_2\ [B_1R_2R_3]_2\ [B_1R_2T_3]_2\ [B_1B_2A_3]_2 \\ [B_1G_2A_3]_2\ [G_1R_2R_3]_2\ [G_1R_2T_3]_2\ [G_1B_2A_3]_2\ [G_1G_2A_3]_2 \end{pmatrix}$
5	23	$\begin{pmatrix} [R_1R_2R_3]_2\ [R_1R_2B_3]_2\ [R_1R_2G_3]_2\ [R_1B_2R_3]_2\ [R_1B_2B_3]_2 \\ [R_1B_2G_3]_2\ [R_1G_2R_3]_2\ [R_1G_2B_3]_2\ [R_1G_2G_3]_2\ [B_1R_2R_3]_2 \\ [B_1R_2B_3]_2\ [B_1R_2G_3]_2\ [B_1B_2R_3]_2\ [B_1B_2T_3]_2\ [B_1G_2R_3]_2 \\ [B_1G_2T_3]_2\ [G_1R_2R_3]_2\ [G_1R_2B_3]_2\ [G_1R_2G_3]_2\ [G_1B_2R_3]_2 \\ [G_1B_2T_3]_2\ [G_1G_2R_3]_2\ [G_1G_2T_3]_2 \end{pmatrix}$
6	27	$\begin{pmatrix} [R_1R_2R_3]_2\ [R_1R_2B_3]_2\ [R_1R_2G_3]_2\ [R_1B_2R_3]_2\ [R_1B_2B_3]_2 \\ [R_1B_2G_3]_2\ [R_1G_2R_3]_2\ [R_1G_2B_3]_2\ [R_1G_2G_3]_2\ [B_1R_2R_3]_2 \\ [B_1R_2B_3]_2\ [B_1R_2G_3]_2\ [B_1B_2R_3]_2\ [B_1B_2B_3]_2\ [B_1B_2G_3]_2 \\ [B_1G_2R_3]_2\ [B_1G_2B_3]_2\ [B_1G_2G_3]_2\ [G_1R_2R_3]_2\ [G_1R_2B_3]_2 \\ [G_1R_2G_3]_2\ [G_1B_2R_3]_2\ [G_1B_2B_3]_2\ [G_1B_2G_3]_2\ [G_1G_2R_3]_2 \\ [G_1G_2B_3]_2\ [G_1G_2G_3]_2 \end{pmatrix}$

Table 5. Computation example for $\Pi(3)$

B Number of cells labelled with 2

n	Number of cells labelled 2 at each configuration (from 0 to 17)																	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1	2	4	7	9													
3	1	2	4	8	15	23	27											
4	1	2	4	8	16	31	53	73	81									
5	1	2	4	8	16	32	63	115	179	227	243							
6	1	2	4	8	16	32	64	127	241	409	585	697	729					
7	1	2	4	8	16	32	64	128	255	495	891	1403	1867	2123	2187			
8	1	2	4	8	16	32	64	128	256	511	1005	1881	3185	4673	5857	6433	6561	
9	1	2	4	8	16	32	64	128	256	512	1023	2027	3891	6947	11043	15203	18147	19427
10	1	2	4	8	16	32	64	128	256	512	1024	2047	4073	7945	14729	24937	37289	48553
11	1	2	4	8	16	32	64	128	256	512	1024	2048	4095	8167	16091	30619	54395	87163
12	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8191	16357	32425	62801	115633
13	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16383	32739	65139	127651
14	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32767	65505	130617
15	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65535	131039
16	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131071
17	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
18	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
19	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072

Table 6. Number of cells labelled 2 at each configuration. Steps 0 to 17.

n	Number of cells labelled 2 at each configuration (from 18 to 25)							
	18	19	20	21	22	23	24	25
2 - 8								
9	19683							
10	55721	58537	59049					
11	123131	152827	169979	176123	177147			
12	195953	297457	399089	475633	516081	529393	531441	
13	241235	427219	689363	994003	1273811	1467347	1561555	1590227
14	257929	496537	909689	1543801	2372729	3261817	4014969	4496249
15	261627	519163	1012395	1902763	3363179	5460331	8007275	10538603
16	262109	523705	1042417	2050721	3927553	7168705	12186689	18927937
17	262143	524251	1047923	2089827	4135555	8028995	15023811	26524099
18	262144	524287	1048537	2096425	4185673	8315209	16300105	31081801
19	262144	524288	1048575	2097111	4193499	8378523	16686555	32930523

Table 7. Number of cells labelled 2 at each configuration. Steps 18 to 25.

n	Number of cells labelled 2 at each configuration						
	26	27	28	29	30	31	32
2 - 12							
13	1594323						
14	4713337	4774777	4782969				
15	12526187	13705835	14201451	14332523	14348907		
16	26553153	33603393	38758209	41613121	42735425	43013953	43046721
17	43301315	64409027	86709699	105964995	119129539	125961667	128484803
18	56571721	96349513	151011657	215527753	279384393	331059529	364220745
19	63682011	118735323	209492955	343710683	517551067	710439899	889828315

Table 8. Number of cells labelled 2 at each configuration. Steps 26 to 32.

n	Number of cells labelled 2 at each configuration					
	33	34	35	36	37	38
2 - 16						
17	129074627	129140163				
18	380408137	386044233	387289417	387420489		
19	1026339803	1108849627	1146860507	1159377883	1161999323	1162261467

Table 9. Number of cells labelled 2 at each configuration. Steps 33 to 38.

C P–Lingua model file

The content of the file in P–Lingua format, containing the specification of the model, is shown below.

```
@model<tissue_psystems>
def main()
{
/* tissue P system skeleton */
call example_tissue(n);
}
def example_tissue(n)
{
call init_cells();
call init_multisets(n);
call init_rules(n);
}
def init_cells()
{
@mu = [[]'2]'0;
}

def init_rules(n)
{
```

```
/* r1 */ [A{i}]'2 --> [R{i}]'2 [T{i}]'2 : 1<=i<=n;
/* r2 */ [T{i}]'2 --> [B{i}]'2 [G{i}]'2 : 1<=i<=n;
}

def init_multisets(n)
{
@ms(2) += A{i} : 1<=i<=n;
}
```

D Daikon integration in MeCoSim - plugin and config files

This appendix presents some technical details about the developed plugin, and the simple process of integration with MeCoSim.

The main code of the program is contained in `DaikonInterface.jar` program, that receives an input file with a compatible format for Daikon invariants detector. However, an additional jar file has been developed, `DaikonPlugin.jar`, to permit selecting among the different extraction files generated by the simulator.

To implement the integration of the program, the only work we have to do is the addition of a few lines in the file `plugins-properties` of MeCoSim, as follows:

```
plugin-daikon = daikonPlugin.Main
pluginname-daikon = Daikon
pluginmethod-daikon = pluginHook
pluginparam-daikon-1 = userfiles/daikon-files.txt
pluginjar-daikon-1 = DaikonInterface.jar
pluginjar-daikon-2 = DaikonPlugin.jar
```

E Algorithms

Algorithm 1 Calculating the number of membranes

```
function TriColor(n)
    $numberCellsStep[0..2n]$
    for i=0 do 2n-1
        $numberCellsStep[i] \leftarrow 0$
    end for
    $a \leftarrow n$
    $t \leftarrow 0$
    $step \leftarrow 0$
    stack.Push($a, t, step$)
    $numberCellsStep[step] \leftarrow numberCellsStep[step] + 1$
    while not stack.IsEmpty() do
        $a, t, step \leftarrow$ stack.Pop()
        $step \leftarrow step + 1$
        if $t > 1$ then
            $t \leftarrow t - 1$
            stack.Push($a, t, step$)
            stack.Push($a, t, step$)
            $numberCellsStep[step] \leftarrow numberCellsStep[step] + 1$
        else if $t = 1$ then
            if $a > 0$ then
                $t \leftarrow t - 1$
                stack.Push($a, t, step$)
                stack.Push($a, t, step$)
                $numberCellsStep[step] \leftarrow numberCellsStep[step] + 1$
            else
                $numberCellsStep[step] \leftarrow numberCellsStep[step] + 1$
            end if
        else if $a > 0$ then
            stack.Push($a - 1, t, step$)
            stack.Push($a - 1, t + 1, step$)
            $numberCellsStep[step] \leftarrow numberCellsStep[step] + 1$
        end if
    end while
```

```
    for i=1 do 2n-1
        $numberCellsStep[i] \leftarrow numberCellsStep[i - 1] + numberCellsStep[i]$
    end for
    return $numberCellsStep$
end function
```

Algorithm 2 Testing the invariants

```
function Fn2(n)
    return $2 * *(n + 2) - 2 * n - 3$
end function
function Fn3(n)
    return $2 * *(n + 3) - 2 * n * *2 - 4 * n - 7$
end function
function Fn4(n)
    $m1 \leftarrow n * *3 * 4$
    $r1 \leftarrow m1 \mod 3$
    $m1 \leftarrow m1/3$
    $m2 \leftarrow 2 * n * *2$
    $m3 \leftarrow 32 * n$
    $r3 \leftarrow m3 \mod 3$
    $m3 \leftarrow m3/3$
    $r \leftarrow (r1 + r3)/3$
    return $2 * *(n + 4) - m1 - m2 - m3 - r - 15$
end function
function Fn5(n)
    $m1 \leftarrow n * *4 * 2$
    $r1 \leftarrow m1 \mod 3$
    $m1 \leftarrow m1/3$
    $m2 \leftarrow n * *2 * 28$
    $r2 \leftarrow m2 \mod 3$
    $m2 \leftarrow m2/3$
    $m3 \leftarrow 20 * n$
    $r \leftarrow (r1 + r2)/3$
    return $2 * *(n + 5) - m1 - m2 - m3 - r - 31$
end function
```

```
function Fn6(n)
    $m1 \leftarrow n**5*4$
    $r1 \leftarrow m1 \mod 15$
    $m1 \leftarrow m1/15$
    $m2 \leftarrow n**4*2$
    $m2 \leftarrow m2/3$
    $m3 \leftarrow n**3*20$
    $r3 \leftarrow m3 \mod 3$
    $m3 \leftarrow m3/3$
    $m4 \leftarrow n**2*32$
    $r4 \leftarrow m4 \mod 3$
    $m4 \leftarrow m4/3$
    $m5 \leftarrow n*676$
    $r5 \leftarrow m5 \mod 15$
    $m5 \leftarrow m5/15$
    $r \leftarrow (r3+r4)/3$
    $rr \leftarrow (r1+r5)/3$
    return $2**(n+6) - m1 + m2 - m3 - m4 - m5 - r - rr - 63$
end function
function Test(n)
    $a \leftarrow TriColor(n)$
    $nr \leftarrow len(a) - 1$
    $results \leftarrow []$
    if $n+2 \leq nr$ then
        $results.append(('n+2', a[n+2] == Fn2(n), Fn2(n))$
    end if
    if $n+3 \leq nr$ then
        $results.append(('n+3', a[n+3] == Fn3(n), Fn3(n))$
    end if
    if $n+4 \leq nr$ then
        $results.append(('n+4', a[n+4] == Fn4(n), Fn4(n))$
    end if
    if $n+5 \leq nr$ then
        $results.append(('n+5', a[n+5] == Fn5(n), Fn5(n))$
    end if
    if $n+6 \leq nr$ then
        $results.append(('n+6', a[n+6] == Fn6(n), Fn6(n))$
    end if
    return $results$
end function
```

Author Index