RGNC REPORT 1/2011

Ninth Brainstorming Week on Membrane Computing

Sevilla, January 31 – February 4, 2011

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

Editors

Research Group on Natural Computing
REPORTS

UNIVERSIDAD DE SEVILLA

Ninth Brainstorming Week on Membrane Computing

Sevilla, January 31 – February 4, 2011

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

Editors

Ninth Brainstorming Week on Membrane Computing

Sevilla, January 31 – February 4, 2011

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

Editors

RGNC REPORT 1/2011

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2011

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

Preface

This volume contains the papers emerged from the Ninth Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from January 31 to February 4, 2011, in the organization of the Research Group on Natural Computing from the Department of Computer Science and Artificial Intelligence of Sevilla University. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and the next seven editions took place in Sevilla at the beginning of February 2004, 2005, 2006, 2007, 2008, 2009, and 2010, respectively.

In the style of previous meetings in this series, the ninth BWMC was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and on cooperation. Interesting enough, both the number of presentations and the number of participants have continuously increased in the last years. (The list of the participants is given in the end of this preface.) However, in the style of the of this series of meeting, these presentations were "provocative", mainly proposing new ideas, open problems, research topics, results which need further improvements. The efficiency of this type of meetings was again proved to be very high and the present volume proves this assertion.

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

A selection of the papers from this volume will be considered for publication in a special issues of *International Journal of Natural Computing Research*. After the first BWMC, a special issue of *Natural Computing* – volume 2, number 3, 2003, and a special issue of *New Generation Computing* – volume 22, number 4, 2004, were published; papers from the second BWMC have appeared in a special issue of *Journal of Universal Computer Science* – volume 10, number 5, 2004, as well as in a special issue of *Soft Computing* – volume 9, number 5, 2005; a selection

of papers written during the third BWMC have appeared in a special issue of *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006; after the fourth BWMC a special issue of *Theoretical Computer Science* was edited – volume 372, numbers 2-3, 2007; after the fifth edition, a special issue of *International Journal of Unconventional Computing* was edited – volume 5, number 5, 2009; a selection of papers elaborated during the sixth BWMC has appeared in a special issue of *Fundamenta Informaticae* – volume 87, number 1, 2008; after the seventh BWMC, a special issue of *International Journal of Computers, Control and Communication* was published – volume 4, number 3, 2009; finally, a selection of papers elaborated during the eight BWMC was published as a special issue of *Romanian Journal of Information Science and Technology* (published by the Romanian Academy) – volume 13, number 2, 2010. Other papers elaborated during the ninth 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. Arroyo Montoro Fernando, Polytechnical University of Madrid, Spain, `farroyo@eui.upm.es`
2. Cámpora Pérez Daniel Hugo, University of Sevilla, Spain `danielcampora@gmail.com`
3. Carmona Pirez Jonás, Andalusian Center of Development in Biology CABD-CSIC, Seville, Spain `jcarpir@upo.es`
4. Carnero Iglesias Javier, University of Sevilla, Spain `javier@carnero.net`
5. Cecilia Canales José María, University of Murcia, Spain `chema@ditec.um.es`
6. Cavaliere Matteo, Spanish National Biotechnology Centre, Madrid, Spain `mcavaliere@cnb.csic.es`
7. Cienciala Ludek, Silesian University, Opava, Czech Republic `ludek.cienciala@fpf.slu.cz`
8. Ciencialova Lucie, Silesian University, Opava, Czech Republic `lucie.ciencialova@fpf.slu.cz`
9. Csuhaj-Varjú Erzsébet, Hungarian Academy of Sciences, Budapest, Hungary, `csuhaj@sztaki.hu`
10. De-Vega-Rodríguez David, University of Sevilla, Spain, `ddevega@gmail.com`
11. Díaz-Pernil Daniel, University of Sevilla, Spain, `sbdani@us.es`

12. Fondevilla Moreu Cristian, University of Lleida, Spain,
cfondevilla@matematica.udl.cat
13. Franco Giuditta, University of Verona, Italy,
giuditta.franco@univr.it
14. García-Carrasco José Manuel, University of Murcia, Spain
jmgarcia@ditec.um.es
15. García-Quismondo Manuel, University of Sevilla, Spain
mgarciaquismondo@us.es
16. Gazdag Zsolt, Eötvös Loránd University, Budapest, Hungary,
gazdagzs@inf.elte.hu
17. Gheorghe Marian, University of Sheffield, United Kingdom,
marian@dcs.shef.ac.uk
18. Giraldez Crú Jesús, University of Sevilla, Spain
giraldez.jesus@gmail.com
19. Graciani-Díaz Carmen, University of Sevilla, Spain,
cgdiaz@us.es
20. Guisado-Lizar José Luis, University of Sevilla, Spain,
jlguisado@us.es
21. Gutiérrez-Naranjo Miguel Ángel, University of Sevilla, Spain,
magutier@us.es
22. Ionescu Mihai, University of Piteşti, Romania
armandmihai.ionescu@gmail.com
23. Ipate Florentin Eugen, University of Piteşti, Romania,
florentin.ipate@ifsoft.ro
24. Kelemenova Alica, Silesian University, Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz
25. Krassovitskiy Alexander, Rovira i Virgili University, Tarragona, Spain,
alexander.krassovitskiy@estudiants.urv.cat
26. Langer Miroslav, Silesian University, Opava, Czech Republic
miroslav.langer@fpf.slu.cz
27. Lefticaru Raluca, University of Piteşti, Romania,
raluca.lefticaru@gmail.com
28. Leporati Alberto, University of Milano-Bicocca, Italy,
leporati@disco.unimib.it
29. Marchetti Luca, University of Verona, Italy,
luca.marchetti@univr.it
30. Martínez-del-Amor Miguel Angel, University of Sevilla, Spain,
mdelamor@us.es
31. Millán Alejandro, University of Sevilla, Spain,
amillan@us.es
32. Mina-Caicedo Julián Andrés, University of Sevilla, Spain,
julmincai@alum.us.es
33. Molina Abril, Helena, University of Sevilla, Spain,
habril@us.es

34. Murphy Niall, NUI Maynooth, Ireland
nmurphy@cs.nuim.ie
35. Nicolescu Radu, University of Auckland, New Zealand
r.nicolescu@auckland.ac.nz
36. Obtułowicz Adam, Polish Academy of Sciences, Poland,
A.Obtulowicz@impan.gov.pl
37. Păun Gheorghe, Institute of Mathematics of the Romanian Academy, Bucharest, Romania, and University of Sevilla, Spain,
george.paun@imar.ro, gpaun@us.es
38. Peña Camacho Miguel Angel, Polytechnical University of Madrid, Spain,
mapc@eui.upm.es
39. Pérez-Hurtado-de-Mendoza Ignacio, University of Sevilla, Spain,
perezh@us.es
40. Pérez-Jiménez Mario de Jesús, University of Sevilla, Spain,
marper@us.es
41. Porreca Antonio E., University of Milano-Bicocca, Italy,
porreca@disco.unimib.it
42. Quirós-Carmona Juan, University of Sevilla, Spain,
quirole@gmail.com
43. Reina-Molina Raúl, University of Sevilla, Spain,
raulrm75@gmail.com
44. Riscos-Núñez Agustín, University of Sevilla, Spain,
ariscosn@us.es
45. Rodríguez-Patón Aradas Alfonso, Polytechnical University of Madrid, Spain,
arpaton@fi.upm.es
46. Romero-Campero Francisco José, University of Sevilla, Spain,
fran@us.es
47. Rogozhin Yurii, Institute of Mathematics and Computer Science, Chişinău, Moldova,
rogozhin@math.md
48. Romero-Jiménez Álvaro, University of Sevilla, Spain,
romero.alvaro@us.es
49. Rosselló Llompart Francesc, University of Balearic Islands, Spain,
frossello@mac.com
50. Sarrión Morillo Enrique, University of Sevilla, Spain,
esmesm@gmail.com
51. Sempere Luna José María, Polytechnical University of Valencia, Spain,
jsempere@dsic.upv.es
52. Sosik Petr, Silesian University, Opava, Czech Republic
petr.sosik@fpf.slu.cz
53. Ţurcanu Adrian, University of Piteşti, Romania,
adrianturcanu85@yahoo.com
54. Valencia Cabrera Luis, University of Sevilla, Spain,
lvalencia@us.es

55. Vaszil György, Hungarian Academy of Sciences, Budapest, Hungary,
vaszil@sztaki.hu
56. Verlan Serghei, Paris XII University, Créteil, France,
verlan@univ-paris12.fr
57. Viejo Cortés Julián, University of Sevilla, Spain,
julian@dte.us.es

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 Ciencia e Innovación, grant TIN 2009 - 13192, (iii) Red Temática Nacional en Computación Biomolecular y Biocelular, grant TIN 2008 - 04487-E, (iv) IV Plan Propio de la Universidad de Sevilla, (v) Consejería de Innovación, 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 5, 2011)

Contents

Asynchronous P Systems (Draft)

Tudor Bălănescu[1], Radu Nicolescu[2], and Huiling Wu[2]

[1] Department of Computer Science, University of Piteşti,
Târgu din Vale 1, 110040 Piteşti, Romania,
tudor_balanescu@yahoo.com
[2] Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand,
r.nicolescu@auckland.ac.nz, hwu065@aucklanduni.ac.nz

Summary. In this paper, we propose a new approach to fully asynchronous P systems, and a matching complexity measure, both inspired from the field of distributed algorithms. We validate our approach by implementing several well-known distributed depth-first search (DFS) and breadth-first search (BFS) algorithms. Empirical results show that our P algorithms achieve a performance comparable to the standard versions.

Key words: P systems, synchronous, asynchronous, distributed, depth-first search, breadth-first search

1 Introduction

P systems is bio-inspired computational model, based on the way in which chemicals interact and cross cell membranes, introduced by Păun [20]. The essential specification of a P system includes a membrane structure, objects and rules. Cells evolve by applying rules in a non-deterministic and (potentially maximally) parallel manner. These characteristics make P systems a promising candidate as a model for distributed and parallel computing.

The traditional P system model is *synchronous*, i.e. all cells evolution is controlled by a single global clock. P systems with various *asynchronous* features have been investigated by recent research, such as Casiraghi et al. [3], Cavaliere et al. [6, 4, 5], Freund et al. [11], Gutiérrez et al. [12], Kleijn et al. [13], Pan et al. [18], Yuan et al. [24]. Here we are looking for similar but simpler definitions, closer to the definitions used in the field of distributed algorithms [14, 22], which will enable us to consider essential distributed feature, such as fairness, safety, liveness and possibly infinite executions. In our approach, algorithms are non-deterministic, not necessarily constrained to return exactly the same result.

Fully asynchronous P systems are characterized by the absence of any system clock, much less a global one; however, an outside observer may very well use a clock to time the evolutions. Our approach does *not* require any change in the

static descriptions of P systems, only their *evolutions* differ (i.e. the underlying P engine works differently):

- Local rules execution takes *zero* time units (i.e. it occurs instantaneously).
- The message delay is unpredictable, so outgoing objects can arrive at the target cell in *any* number of time units (after being sent).

For the purpose of *time complexity*, the time unit is chosen greater than any message delay, i.e. the delay between sending and receiving a message is any real number in the closed interval $[0, 1]$.

This paper is organized as follows. Section 2 gives a definition of a simple P module, as a unified model of various P systems. Section 3 presents asynchronous P systems and discusses a standard set of time complexity measures. Section 4 and Section 5 discuss several well-known distributed DFS and BFS algorithms and propose corresponding asynchronous P system implementations. Section 6 compares the complexity of our asynchronous P system algorithms with the theoretical complexity of distributed DFS and BFS algorithms. Finally, Section 7 summarizes our work and highlights future work.

2 Preliminary

In this paper, we use *simple P modules*, an umbrella concept, which is general enough to cover several basic P system families, with *states*, *priorities*, *promoters* and *duplex* channels. For the full definition of P modules and modular compositions, we refer readers to [10].

Essentially, a simple P module is a system, $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_n, \delta)$, where:

1. O is a finite non-empty alphabet of *objects*;
2. $\sigma_1, \ldots, \sigma_n$ are cells, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$, $1 \leq i \leq n$, where:
   - Q_i is a finite set of *states*;
   - $s_{i,0} \in Q_i$ is the *initial state*;
   - $w_{i,0} \in O^*$ is the *initial multiset* of objects;
   - R_i is a finite *ordered* set of rewriting/communication *rules* of the form: $s\ x \rightarrow_\alpha s'\ x'\ (y)_\beta\ |_z$, where: $s, s' \in Q_i$, $x, x', y, z \in O^*$, $\alpha \in \{min, max\}$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$.
3. δ is a set of *digraph* arcs on $\{1, 2, \ldots, n\}$, without reflexive arcs, representing *duplex* channels between cells.

The membrane structure is a digraph with duplex channels, so parents can send messages to children *and* children to parents. Rules are prioritized and are applied in *weak priority* order [19]. The general form of a rule, which transforms state s to state s', is $s\ x \rightarrow_\alpha s'\ x'\ (y)_{\beta_\gamma}\ |_z$. This rule consumes multiset x, and then (after all applicable rules have consumed their left-hand objects) produces multiset x', in the same cell ("*here*"). Also, it produces multiset y and sends it, by *replication*

("*repl*" mode), to all parents ("*up*"), to all children ("*down*") or to all parents and children ("*up and down*"), according to the target indicator $\beta \in \{\uparrow, \downarrow, \updownarrow\}$.

We also use a targeted sending, $\beta = \uparrow_j, \downarrow_j, \updownarrow_j$, where j is either an arc label or a cell ID. If j is an arc label, y is sent via the arc labelled j, provided that it points, respectively, up (to a parent), down (to a child) or in any direction (to either a parent or a child). If j is a cell ID of a structural neighbor, y is sent to that neighbor j, provided that it lies, respectively, up (j is a parent), down (j is a child) or in any direction (j is either a parent or a child); nothing is sent if cell j is not a structural neighbor (we do not use teleportation). More about cell IDs in a following paragraph.

$\alpha \in \{min, max\}$ describes the rewriting mode. In the *minimal* mode, an applicable rule is applied once. In the *maximal* mode, an applicable rule is used as many times as possible and all rules with the same states s and s' can be applied in the maximally parallel manner. Finally, the optional z indicates a multiset of promoters, which enable rules but are not consumed.

Note

The algorithms presented in this paper make full use of duplex channels and work regardless of specific arc orientation. Therefore, to avoid superfluous details, the structure of our sample P systems will be given as undirected graphs, with the assumption that the results will be the same, regardless of actual arc orientation.

Extensions

In this article, we use an extended version of the basic P module framework, described above. Specifically, we assume that each cell $\sigma_i \in K$ was "blessed" from factory with a unique *cell ID* symbol ι_i, which is exclusively used as an *immutable promoter*. We also allow high-level rules, with a simple form of *complex symbols* and *free variable* matching.

To explain these additional features, consider, for example, rule 3.1 of algorithm 2: $S_3\ a\ n_j \rightarrow_{\texttt{min}} S_4\ a\ (c_i) \downarrow_j\ |\iota_i$. This rule uses complex symbols n_j and c_i, where j and i are free variables, which, in principle, could match anything, but, in this case, they will be only required to match cell IDs. Briefly, this rule, promoted by ι_i, consumes one a and one n_j, produces another a and sends down a c_i, where i is the index of the current cell, to child j, if this child exists.

3 Asynchronous P Systems

In traditional P systems, a universal clock is assumed to control the application of all rules, i.e. traditional P systems work synchronously, in *lock-step*. Practically, such universal clock is unrealistic in many distributed computing applications, where there is no such global clock and the communication delay is unpredictable.

Thus, it is interesting to investigate P systems that work in the asynchronous mode.

We define asynchronous P systems as follows. The rule format of asynchronous P systems is the same as for synchronous P systems, i.e., $s\ x \rightarrow_{\alpha} s'\ x'\ (y)_{\beta_\gamma}\ |_z$. However, we focus on typical distributed systems, where communications take substantially longer than actual local computations, therefore we consider that the message delay is totally unpredictable. In such systems, we assume that rules are applied in zero time and each message arrives in its own time t, $t \in [0, 1]$. Synchronous P systems are a special case of asynchronous P systems, where $t = 1$, for all evolutions. The *runtime complexity* of an asynchronous system is the *supremum* over all possible executions. We typically assume that messages sent over the same arc arrive in FIFO order (queue), or, as a possible extension—all messages sent over the same arc eventually arrive, but in arbitrary order (multiset).

We illustrate these concepts by means of a basic algorithm, **Echo** [22], in two distributed scenarios: (1) synchronous and (2) asynchronous, with a different (and less expected) evolution. Essentially, the Echo algorithm starts from a source cell, which broadcasts forward messages. These forward messages transitively reach all cells and, at the end, are reflected back to the initial source. The forward phase establishes a *virtual spanning tree* and the return phase is supposed to follow up its branches. The tree is only virtual, because it does not involve any structural changes; instead, virtual child-parent links are established by way of pointer objects.

Scenario 1 in Figure 1 assumes that all messages arrive in one time unit, i.e. in the synchronous mode. The forward and return phases take the same time, i.e. D time units each, where D is diameter of the undirected graph, G. Scenario 2 in Figure 2 assumes that some messages travel much faster than others, which is bad, but possible in asynchronous mode: $t = \epsilon$, where $0 < \epsilon \ll 1$. In this case, the forward and return phases take very different times, D and $N - 1$ time units, respectively, where N is the number of nodes of the undirected graph, G. The P system rules of the Echo algorithm are presented in Section 5.3.

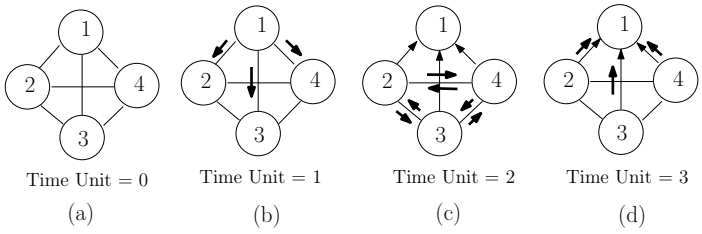


Fig. 1. Echo algorithm in synchronous mode—or in a "lucky" asynchronous mode, when all messages are propagated with the same delay (1). Arcs with arrows indicate child-parent arcs in the virtual spanning tree built by the algorithm. Thick arrows near arcs indicate messages.

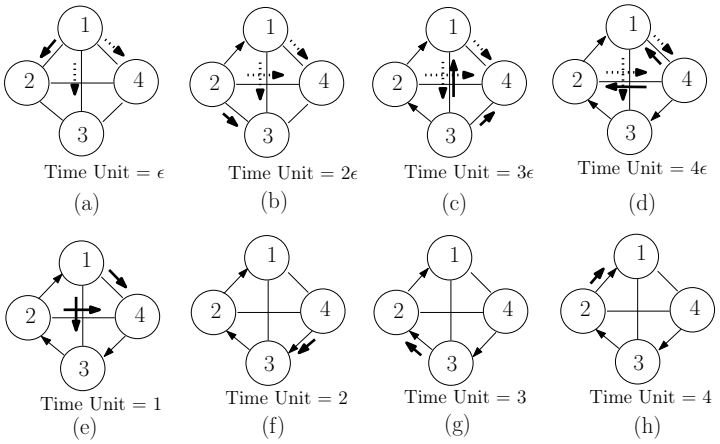


Fig. 2. Echo algorithm in asynchronous mode—one possible "bad" execution, among the many possible. Dotted thick arrows near arcs indicate messages still in transit.

4 Distributed Depth-First Search (DFS)

Depth-first search (DFS) and breadth-first search (BFS) are graph traversal algorithms, which construct a DFS spanning tree and a BFS spanning tree, respectively. Figure 3 shows the structure of a sample P system, Π, based on an "undirected" graph, G, and one possible *virtual* DFS spanning tree, T. We use quotation marks to indicate that G actually is a *directed* graph, but we do not care about arc orientation. The spanning tree is virtual, as it is described by "soft" pointer objects, not by "hard" structural arcs.

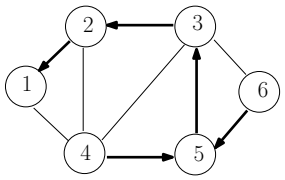


Fig. 3. P system Π based on an "undirected" graph and one possible virtual DFS spanning tree. Thick arrows indicate virtual child-parent arcs in this tree, linked by pointer objects.

DFS is a fundamental technique, inherently *sequential*, or so it appears. Several distributed DFS algorithms have been proposed, which attempt to make DFS run faster on distributed systems, such as the classical DFS [22], Awerbuch's DFS [1], Cidon's DFS [7], Sharma et al's DFS [21], Makki et al's DFS [15], Sense of Direction (SOD) DFS [22]. This is vast topic, which is impossible to present here

at the required length. Therefore, we refer the reader to the original articles, or to a fundamental text, which covers all these algorithms, [22].

Several articles have proposed various synchronous P algorithms for DFS. Gutiérrez-Naranjo et al. proposed a DFS algorithm [12], using inhibitors to avoid visiting already-visited neighbor cells. Dinneen et al. [8] proposed a P algorithm to find disjoint paths in a digraph, using a distributed DFS strategy, which avoids visiting already-visited cells by changing the state of visited cells [9]. Bernardini et al. proposed a DFS algorithm in the P system synchronization problem [2]. This approach uses an operator, $mark_+$, to select one not-yet-visited cell, indicated by a 0 polarity, and then mark the cell as visited, by changing the polarity to +. In this case, the cell that performs a $mark_+$ operation, actually "knows" which child cell has been visited or not, without any message exchanges. In fact, all above mentioned P algorithms implement the classical DFS, which is discussed later in Section 4.2.

In the following sections, we present asynchronous P system implementations of the well-known distributed DFS algorithms, which leverage the parallel and distributed characteristics of P systems.

4.1 Discovering Neighbors

All our distributed DFS and BFS P algorithms, except the SoD algorithm, can, if needed, start with the same preliminary Phase I, in which cells discover their neighbors, i.e. their local topology. Nicolescu et al. have developed P algorithms to discover local topology and local neighbors [16, 9]. In this paper, we propose a crisper algorithm, Algorithm 1, with fewer symbols.

Algorithm 1 (Discovering cell neighbors)

Input: All cells start in the same initial state, S_0, with the same set of rules. Initially, each cell, σ_i, contains a cell ID object, ι_i, which is *immutable* and used as a *promoter*. Additionally, the source cell, σ_s, is decorated with one object a.

Output: All cells end in the same state, S_3. On completion, each cell contains the cell ID object, ι_i, and objects n_j, pointing to their neighbors. The source cell, σ_s, is still decorated with object a. Table 1 shows the neighborhoods of Figure 3, computed by Algorithm 1, in three P steps.

Table 1. Partial Trace of Algorithm 1 for Figure 3.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_0\ \iota_1 a$	$S_0\ \iota_2$	$S_0\ \iota_3$	$S_0\ \iota_4$	$S_0\ \iota_5$	$S_0\ \iota_6$
3	$S_3\ \iota_1 a n_2 n_4$	$S_3\ \iota_2 n_1 n_3 n_4$	$S_3\ \iota_3 n_2 n_4 n_5 n_6$	$S_3\ \iota_4 n_1 n_2 n_3 n_5$	$S_3\ \iota_5 n_3 n_4 n_6$	$S_3\ \iota_6 n_3 n_5$

0. Rules in state S_0:
   1 $S_0\ a \rightarrow_{\texttt{min}} S_1\ ay\ (z) \updownarrow$
   2 $S_0\ z \rightarrow_{\texttt{min}} S_1\ y\ (z) \updownarrow$
   3 $S_0\ z \rightarrow_{\texttt{max}} S_1$
1. Rules in state S_1:
   1 $S_1\ y \rightarrow_{\texttt{min}} S_2\ (n_i) \updownarrow |_{\iota_i}$
   2 $S_1\ z \rightarrow_{\texttt{max}} S_2$
2. Rules for state S_2:
   1 $S_2 \rightarrow_{\texttt{min}} S_3$
   2 $S_2\ z \rightarrow_{\texttt{max}} S_3$

In state S_0, the source cell, σ_s, which is decorated by object a, broadcasts signal z, to all cells, and enters state S_1. Each cell receiving z produces one object y, and changes to state S_1. Superfluous signals z are discarded. Then, in state S_1, each cell that has object y, sends its own ID, which appears as subscript in complex object n_i, to all its neighbors. In state S_2, cells accumulate the received neighbor objects, discard superfluous objects z, and enter S_3.

4.2 Classical DFS

The classical DFS algorithm is based on Tarry's traversal algorithm, which traverses all arcs *sequentially*, in both directions, using a visiting *token* [22]. Because it traverses all arcs twice, serially, the classical DFS algorithm is not the most efficient distributed DFS algorithm.

Algorithm 2 (Classical DFS)

Input: All cells start in the same quiescent state, S_3, and with the same set of rules. Each cell, σ_i, contains an immutable cell ID object, ι_i. All cells know their neighbors, i.e. they have topological awareness, which are indicated by pointer objects, n_j (as built by Algorithm 1). The source cell, σ_s, is additionally decorated with one object, a, which triggers the search.

Output: All cells end in the same final state (S_5). On completion, the cell IDs are intact. Cell σ_s is still decorated with one a and all other cells contain DFS spanning tree pointer objects, indicating predecessors, p_j.

Table 2 shows one possible DFS spanning tree, built by this algorithm, for the P system Π of Figure 3.

Table 2. Partial Trace of Algorithm 2 for Figure 3.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_3\ \iota_1 a n_2 n_4$	$S_3\ \iota_2 n_1 n_3 n_4$	$S_3\ \iota_3 n_2 n_4 n_5 n_6$	$S_3\ \iota_4 n_1 n_2 n_3 n_5$	$S_3\ \iota_5 n_3 n_4 n_6$	$S_3\ \iota_6 n_3 n_5$
19	$S_5\ \iota_1 a$	$S_5\ \iota_2 p_1$	$S_5\ \iota_3 p_2$	$S_5\ \iota_4 p_5$	$S_5\ \iota_5 p_3$	$S_5\ \iota_6 p_5$

3. Rules in state S_3:
   1 $S_3\ an_j \rightarrow_{\texttt{min}} S_4\ a\ (c_i) \downarrow_j |_{\iota_i}$
   2 $S_3\ c_j n_j n_k \rightarrow_{\texttt{min}} S_4\ p_j\ (c_i) \downarrow_k |_{\iota_i}$
4. Rules for state S_4:
   1 $S_4\ c_j n_j \rightarrow_{\texttt{min}} S_4\ (x_i) \downarrow_j |_{\iota_i}$
   2 $S_4\ x_j n_k \rightarrow_{\texttt{min}} S_4\ (c_i) \downarrow_k |_{\iota_i}$
   3 $S_4\ x_j p_k \rightarrow_{\texttt{min}} S_5\ p_k\ (x_i) \downarrow_k |_{\iota_i}$
   4 $S_4\ x_j \rightarrow_{\texttt{min}} S_5$

4.3 Awerbuch DFS

Awerbuch's algorithm [1] and other more efficient algorithms improve time complexity by having the visiting token traversing tree arcs only, all other arcs are traversed in parallel, by auxiliary messages. Specifically, in Awerbuch's algorithm, when the node is visited for the first time, it *notifies* all neighbors that it has been visited and waits until it receives all neighbors' *acknowledgments*. After that, the node can visit one of its unvisited neighbors. Thus, the node knows exactly which of its neighbors have been visited and avoids visiting the already-visited neighbors, which saves time.

Algorithm 3 (Awerbuch DFS)

Input: Same as in Algorithm 2.

Output: Similar to Algorithm 2, but the final state is S_7. Also, cells may contain "garbage" objects, which can be cleared, by using a few more steps.

Table 3 shows the resulting DFS spanning tree, for Figure 3. Table 16 from Appendix A contains full traces for this algorithm, including the preliminary phase, of Algorithm 1.

Table 3. Partial Trace of Algorithm 3 for Figure 3.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_3\ \iota_1 a n_2 n_4$	$S_3\ \iota_2 n_1 n_3 n_4$	$S_3\ \iota_3 n_2 n_4 n_5 n_6$	$S_3\ \iota_4 n_1 n_2 n_3 n_5$	$S_3\ \iota_5 n_3 n_4 n_6$	$S_3\ \iota_6 n_3 n_5$
24	$S_7\ \iota_1 a\ \ldots$	$S_7\ \iota_2 p_1\ \ldots$	$S_7\ \iota_3 p_2\ \ldots$	$S_7\ \iota_4 p_5\ \ldots$	$S_7\ \iota_5 p_3\ \ldots$	$S_7\ \iota_6 p_5\ \ldots$

3. Rules in state S_3:
   1 $S_3\ n_j \rightarrow_{\texttt{min}} S_4\ n_j m_j$
4. Rules in state S_4:
   1 $S_4\ v_j \rightarrow_{\texttt{min}} S_4\ u_j\ (b_i) \downarrow_j |_{\iota_i}$
   2 $S_4\ n_j \rightarrow_{\texttt{min}} S_5\ n_j\ (v_i) \downarrow_j |_{a\iota_i}$
   3 $S_4\ c_j m_j n_j \rightarrow_{\texttt{min}} S_5\ p_j$
   4 $S_4\ n_j \rightarrow_{\texttt{min}} S_5\ n_j\ (v_i) \downarrow_j |_{t\iota_i}$
5. Rules for state S_5:
   1 $S_5\ n_j \rightarrow_{\texttt{min}} S_6\ n_j w_j$
6. Rules for state S_6:
   1 $S_6\ w_j \rightarrow_{\texttt{min}} S_7\ |_{b_j}$
   2 $S_6\ w_j p_k \rightarrow_{\texttt{min}} S_7\ w_j p_k |_{b_l}$
   3 $S_6\ b_j \rightarrow_{\texttt{min}} S_7$
   4 $S_6\ u_j m_j \rightarrow_{\texttt{min}} S_7\ u_j$
   5 $S_6\ a m_j \rightarrow_{\texttt{min}} S_7\ a u_j\ (c_i t) \downarrow_j |_{\iota_i}$
   6 $S_6\ p_k m_j \rightarrow_{\texttt{min}} S_7\ p_k u_j\ (c_i t) \downarrow_j |_{\iota_i}$
   7 $S_6\ p_j \rightarrow_{\texttt{min}} S_7\ p_j\ (x_i t) \downarrow_j |_{\iota_i}$
   8 $S_6\ t \rightarrow_{\texttt{min}} S_7$
7. Rules for state S_7:

1 $S_7\ w_j \rightarrow_{\texttt{min}} S_7\ |_{b_j}$
2 $S_7\ w_j p_k \rightarrow_{\texttt{min}} S_7\ w_j p_k$
3 $S_7\ p_k m_j \rightarrow_{\texttt{min}}$
$S_7\ p_k u_j\ (c_i t) \downarrow_j |_{b_l \iota_i}$
4 $S_7\ p_j \rightarrow_{\texttt{min}} S_7\ p_j\ (x_i t) \downarrow_j |_{b_l \iota_i}$
5 $S_7\ b_j \rightarrow_{\texttt{min}} S_7$
6 $S_7\ m_k x_j \rightarrow_{\texttt{min}} S_7\ u_k\ (c_i t) \downarrow_k |_{\iota_i}$
7 $S_7\ p_k x_j \rightarrow_{\texttt{min}} S_7\ p_k\ (x_i t) \downarrow_k |_{\iota_i}$
8 $S_7\ v_j \rightarrow_{\texttt{min}} S_7\ u_j\ (b_i) \downarrow_j |_{\iota_i}$
9 $S_7\ u_j m_j \rightarrow_{\texttt{min}} S_7\ u_j$
10 $S_7\ a x_j \rightarrow_{\texttt{min}} S_7\ a$
11 $S_7\ t \rightarrow_{\texttt{min}} S_7$

4.4 Cidon DFS

Cidon's algorithm [7] improves Awerbuch's algorithm by not using *acknowledgments*, therefore removing a delay. The token holding cell does not wait for the neighbors' acknowledgments, but immediately visits a neighbor. However, it needs to record the most recent neighbor used, to solve cases when visiting *notifications* arrive after the visiting *token*.

Algorithm 4 (Cidon DFS)

Input: Same as in Algorithm 2.

Output: Similar to Algorithm 2, but the final state is S_5. Also, cells may contain "garbage" objects, which can be cleared, by using a few more steps.

Table 4 shows one possible DFS spanning tree, built by this algorithm, for the P system Π of Figure 3.

Table 4. Partial Trace of Algorithm 4 for Figure 3.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_3\ \iota_1 a n_2 n_4$	$S_3\ \iota_2 n_1 n_3 n_4$	$S_3\ \iota_3 n_2 n_4 n_5 n_6$	$S_3\ \iota_4 n_1 n_2 n_3 n_5$	$S_3\ \iota_5 n_3 n_4 n_6$	$S_3\ \iota_6 n_3 n_5$
12	$S_5\ \iota_1 a\ \ldots$	$S_5\ \iota_2 p_1\ \ldots$	$S_5\ \iota_3 p_2\ \ldots$	$S_5\ \iota_4 p_5\ \ldots$	$S_5\ \iota_5 p_3\ \ldots$	$S_5\ \iota_6 p_5\ \ldots$

3. Rules in state S_3:
   1 $S_3\ n_j \rightarrow_{\texttt{min}} S_4\ n_j m_j$
   2 $S_3\ a \rightarrow_{\texttt{min}} S_4\ at$
4. Rules in state S_4:
   1 $S_4\ a n_j m_j \rightarrow_{\texttt{min}}$
   $S_5\ a v_j\ (v_i c_i t) \downarrow_j |_{t \iota_i}$
   2 $S_4\ c_k n_k m_k n_j m_j \rightarrow_{\texttt{min}}$
   $S_5\ p_k r_j m_j\ (v_i c_i t) \downarrow_j |_{t \iota_i}$
   3 $S_4\ c_k m_k n_j m_j \rightarrow_{\texttt{min}}$
   $S_5\ p_k r_j m_j\ (v_i c_i t) \downarrow_j |_{t \iota_i}$
   4 $S_4\ c_j n_j m_j \rightarrow_{\texttt{min}} S_5\ p_j\ (x_i t) \downarrow_j |_{t \iota_i}$
   5 $S_4\ c_j m_j \rightarrow_{\texttt{min}} S_5\ p_j\ (x_i t) \downarrow_j |_{t \iota_i}$
   6 $S_4\ m_j \rightarrow_{\texttt{min}} S_5\ m_j\ (v_i) \downarrow_j |_{t \iota_i}$
   7 $S_4\ v_j n_j \rightarrow_{\texttt{min}} S_4\ v_j$
   8 $S_4\ t \rightarrow_{\texttt{min}} S_5$
5. Rules for state S_5:
   1 $S_5\ r_k v_k n_j \rightarrow_{\texttt{min}} S_5\ r_j\ (c_i t) \downarrow_j |_{\iota_i}$
   2 $S_5\ r_k v_k p_j \rightarrow_{\texttt{min}} S_5\ p_j\ (x_i t) \downarrow_j |_{\iota_i}$
   3 $S_5\ x_j n_k m_k \rightarrow_{\texttt{min}}$
   $S_5\ r_k m_k\ (v_i c_i t) \downarrow_k |_{t \iota_i}$
   4 $S_5\ x_j p_k r_j \rightarrow_{\texttt{min}}$
   $S_5\ p_k r_j\ (x_i t) \downarrow_k |_{t \iota_i}$
   5 $S_5\ c_j p_k \rightarrow_{\texttt{min}} S_5\ p_k v_j$
   6 $S_5\ v_j n_j \rightarrow_{\texttt{min}} S_5\ v_j$
   7 $S_5\ a x_j \rightarrow_{\texttt{min}} S_5\ a$
   8 $S_5\ t \rightarrow_{\texttt{min}} S_5$

4.5 Sharma DFS

Sharma et al.'s algorithm [21] further improves time complexity, at the cost of increasing the message size, by including a *list* of visited nodes when passing the visiting *token* [23]. Thus, it eliminates unnecessary message exchanges to inform neighbors of visited status.

Algorithm 5 (Sharma DFS)

Input: Same as in Algorithm 2.

Output: Similar to Algorithm 2, but the final state is S_4. Also, cells may contain "garbage" objects, which can be cleared, by using a few more steps.

Table 5 shows one possible DFS spanning tree, built by this algorithm, for the P system Π of Figure 3.

Table 5. Partial Trace of Algorithm 5 for Figure 3.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_3\ \iota_1 a n_2 n_4$	$S_3\ \iota_2 n_1 n_3 n_4$	$S_3\ \iota_3 n_2 n_4 n_5 n_6$	$S_3\ \iota_4 n_1 n_2 n_3 n_5$	$S_3\ \iota_5 n_3 n_4 n_6$	$S_3\ \iota_6 n_3 n_5$
11	$S_4\ \iota_1 a\ \ldots$	$S_4\ \iota_2 p_1\ \ldots$	$S_4\ \iota_3 p_2\ \ldots$	$S_4\ \iota_4 p_5\ \ldots$	$S_4\ \iota_5 p_3\ \ldots$	$S_4\ \iota_6 p_5\ \ldots$

3. Rules in state S_3:
   1 $S_3\ a n_j \rightarrow_{\texttt{min}} S_4\ a\ (c_i v_i t) \downarrow_j |_{\iota_i}$
   2 $S_3\ n_j \rightarrow_{\texttt{min}} S_4\ |_{t v_j}$
   3 $S_3\ c_j \rightarrow_{\texttt{min}} S_4\ p_j\ (c_i v_i t) \downarrow_k |_{n_k \iota_i}$
   4 $S_3\ c_j \rightarrow_{\texttt{min}} S_4\ p_j\ (x_i v_i v_j t) \downarrow_j |_{\iota_i}$
   5 $S_3\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t n_k}$
   6 $S_3\ t \rightarrow_{\texttt{min}} S_4$

4. Rules for state S_4:
   1 $S_4\ n_j \rightarrow_{\texttt{min}} S_4\ v_j$
   2 $S_4\ x_j \rightarrow_{\texttt{min}} S_4\ (c_i v_i t) \downarrow_k |_{n_k \iota_i}$
   3 $S_4\ x_j \rightarrow_{\texttt{min}} S_4\ (x_i v_i t) \downarrow_k |_{p_k \iota_i}$
   4 $S_4\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t n_k}$
   5 $S_4\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t p_k}$
   6 $S_4\ t \rightarrow_{\texttt{min}} S_4$
   7 $S_4\ a x_j \rightarrow_{\texttt{min}} S_4\ a$

4.6 Makki DFS

Makki et al.'s algorithm [15] improves Sharma et al.'s algorithm by using a *dynamic backtracking* technique. It keeps track of the most recent *split point*, i.e. the lowest ancestor node. When the search path backtracks to a node, if the node has a non-tree edge to its split point, it backtracks to the split point directly via that edge, rather than following the longer tree path to its split point.

Algorithm 6 (Makki DFS)

Input: Same as in Algorithm 2.

Output: Similar to Algorithm 2, but the final state is S_4. Also, cells may contain "garbage" objects, which can be cleared, by using a few more steps.

Table 6 shows one possible DFS spanning tree, built by this algorithm, for the P system Π of Figure 3.

Table 6. Partial Trace of Algorithm 6 for Figure 3.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_3\ \iota_1 a n_2 n_4$	$S_3\ \iota_2 n_1 n_3 n_4$	$S_3\ \iota_3 n_2 n_4 n_5 n_6$	$S_3\ \iota_4 n_1 n_2 n_3 n_5$	$S_3\ \iota_5 n_3 n_4 n_6$	$S_3\ \iota_6 n_3 n_5$
10	$S_4\ \iota_1 a\ \ldots$	$S_4\ \iota_2 p_1\ \ldots$	$S_4\ \iota_3 p_2\ \ldots$	$S_4\ \iota_4 p_5\ \ldots$	$S_4\ \iota_5 p_3\ \ldots$	$S_4\ \iota_6 p_5\ \ldots$

3. Rules in state S_3:
   1 $S_3\ a n_j \rightarrow_{\texttt{min}} S_4\ a\ (c_i v_i s_i t) \downarrow_j |_{\iota_i}$
   2 $S_3\ n_j \rightarrow_{\texttt{min}} S_4\ |_{t v_j}$
   3 $S_3\ c_j s_m \rightarrow_{\texttt{min}}$
   $S_4\ p_j r_m\ (c_i v_i s_i t) \downarrow_k |_{n_k n_l \iota_i}$
   4 $S_3\ c_j s_l \rightarrow_{\texttt{min}}$
   $S_4\ p_j r_l\ (c_i v_i s_l t) \downarrow_k |_{n_k \iota_i}$
   5 $S_3\ c_j \rightarrow_{\texttt{min}} S_4\ p_j r_k\ (x_i v_i t) \downarrow_k |_{s_k \iota_i}$
   6 $S_3\ c_j \rightarrow_{\texttt{min}} S_4\ p_j r_k\ (x_i v_i t) \downarrow_j |_{s_k \iota_i}$
   7 $S_3\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t n_k}$
   8 $S_3\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t s_k}$
   9 $S_3\ t \rightarrow_{\texttt{min}} S_4$

4. Rules for state S_4:
   1 $S_4\ n_j \rightarrow_{\texttt{min}} S_4\ v_j$
   2 $S_4\ x_j \rightarrow_{\texttt{min}} S_4\ (c_i v_i s_i t) \downarrow_k |_{n_k n_l \iota_i}$
   3 $S_4\ x_j r_l \rightarrow_{\texttt{min}}$
   $S_4\ (c_i v_i s_i s_l t) \downarrow_k |_{n_k \iota_i}$
   4 $S_4\ x_j \rightarrow_{\texttt{min}} S_4\ (x_i v_i t) \downarrow_k |_{r_k \iota_i}$
   5 $S_4\ x_j \rightarrow_{\texttt{min}} S_4\ (x_i v_i t) \downarrow_k |_{p_k \iota_i}$
   6 $S_4\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t n_k}$
   7 $S_4\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t r_k}$
   8 $S_4\ v_j \rightarrow_{\texttt{min}} S_4\ v_j\ (v_j) \downarrow_k |_{t p_k}$
   9 $S_4\ t \rightarrow_{\texttt{min}} S_4$
   10 $S_4\ a x_j \rightarrow_{\texttt{min}} S_4\ a$

4.7 Sense of Direction DFS

With Sense of Direction (SOD), the node labeling is not required. Instead, arc labeling is used, with the following properties:

- Edges are labeled with elements of a group G, typically $G = Z_n$, where $Z_n = \{0, 1, \ldots, n-1\}$.
- Given labeled arcs $a_0 \xrightarrow{x_1} a_1, a_1 \xrightarrow{x_2} a_2, \ldots a_{k-1} \xrightarrow{x_k} a_k$, the path $a_0 \xrightarrow{x_1} a_1 \xrightarrow{x_2} a_2 \ldots a_{k-1} \xrightarrow{x_k} a_k$ has label $x_1 + x_2 + \ldots + x_k$.
- Given labelled paths $a \overset{x}{\Rightarrow} b$ and $c \overset{x}{\Rightarrow} d$, $a = c$ if and only if $b = d$.

Thus, in search algorithms, path labels can very handily indicate the already-visited nodes. Path labels are kept as a growing list and are appended when the search path passes a node.

If the search path reaching the node, a_k, wants to visit the node, a_{k+1}, it first checks whether a_{k+1} is an already-visited node, e.g., a_i, $0 \leq i \leq n$. The node a_k checks whether one of the partial path labels, e.g., $x_{i+1} + \ldots + x_k + x_{k+1}$, equals zero. If yes, then $a_{k+1} = a_i$, thus a_{k+1} is an already-visited node. We refer the readers to [22] for more details about SOD.

Figure 4 shows a sample P system based on directed graph with SOD arc labels.

Algorithm 7 (Sense of Direction DFS)

For this particular algorithm, here, we only present a P system-like high-level pseudo-code. Additional investigation is required to achieve an efficient translation to usual rewriting rules.

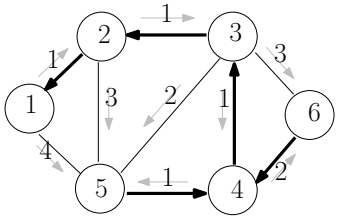


Fig. 4. A sample P system based on a SOD structure, with arc labelling, indicated by gray arrows. Thick arc arrows indicate a possible virtual DFS tree.

Input: All cells start with the same set of rules and start in the same quiescent state, S_0. Initially, all cells contain objects indicating the labels of neighbor arcs: objects o_j for outgoing arcs and objects e_j for incoming arcs. The source cell, σ_s, is additionally decorated with one trigger object, a.

Output: All cells end in the same final state, S_1. On completion, cell σ_s is still decorated with one a. All other cells contain DFS spanning tree pointer objects, indicating its tree predecessors: p_j, for incoming arcs and q_j, for outgoing arcs. Also, cells may contain "garbage" objects, which can be cleared, in a few more steps.

Table 7 shows one possible DFS spanning tree, built by this algorithm, for the P system of Figure 4.

Table 7. Partial Trace of Algorithm 7 for Figure 4.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_0 a o_1 o_4$	$S_0 e_1 o_1 o_3$	$S_0 e_1 o_1 o_2 o_3$	$S_0 e_1 o_1 o_2$	$S_0 e_1 e_2 e_3 e_4$	$S_0 e_2 e_3$
11	$S_1\ a\ \ldots$	$S_1\ p_1\ \ldots$	$S_1\ p_1\ \ldots$	$S_1\ p_1\ \ldots$	$S_1\ p_1\ \ldots$	$S_1\ p_2\ \ldots$

The ruleset below uses a few additional "magical" algebraic operators and prompters, which do fit properly into the basic framework outlined in Section 2 (or not yet).

- Operation $\pi \oplus j$ adds j, modulo n, to every element in list π and also appends $+j$ to list π.
- Operation $\pi \ominus j$ subtracts j, modulo n, from every element in list π and also appends $n - j$ (i.e. $-j$ modulo n) to list π.
- Complex promoters $\pi \oplus j?$ and $\pi \ominus j?$ enable the associated rule only if the resulting list does not contain any 0.

0. Rules in state S_0:
   1 $S_0\ a o_j \to_{\texttt{min}} S_1\ a\ (c_j b_{\oplus j}) \uparrow_j$
   2 $S_0\ b_\pi o_j c_k e_k \to_{\texttt{min}}$
   $S_1\ p_k(c_j b_{\pi \oplus j}) \uparrow_j |_{\pi \oplus j?}$
   3 $S_0\ b_\pi e_j c_k e_k \to_{\texttt{min}}$
   $S_1\ p_k(l_j b_{\pi \ominus j}) \downarrow_j |_{\pi \ominus j?}$
   4 $S_0\ b_\pi o_j l_k o_k \to_{\texttt{min}}$
   $S_1\ q_k(c_j b_{\pi \oplus j}) \uparrow_j |_{\pi \oplus j?}$

5 $S_0\ b_\pi e_j l_k o_k \rightarrow_{\texttt{min}}$
$S_1\ q_k(l_j b_{\pi\ominus j})\downarrow_j |_{\pi\ominus j?}$
6 $S_0\ b_\pi c_j e_j \rightarrow_{\texttt{min}} S_1\ p_j(x_j b_{\pi\ominus j})\downarrow_j$
7 $S_0\ b_\pi l_j o_j \rightarrow_{\texttt{min}} S_1\ p_j(x_j b_{\pi\oplus j})\uparrow_j$

1. Rules in state S_1:

1 $S_1\ b_\pi x_k o_j \rightarrow_{\texttt{min}}$
$S_1\ (c_j b_{\pi\oplus j})\uparrow_j |_{\pi\oplus j?}$
2 $S_1\ b_\pi x_k e_j \rightarrow_{\texttt{min}}$
$S_1\ (l_j b_{\pi\ominus j})\downarrow_j |_{\pi\ominus j?}$
3 $S_1\ b_\pi x_k p_j \rightarrow_{\texttt{min}} S_1\ p_j\ (x_j b_{\pi\ominus j})\downarrow_j$
4 $S_1\ b_\pi x_k q_j \rightarrow_{\texttt{min}} S_1\ q_j\ (x_j b_{\pi\oplus j})\uparrow_j$
5 $S_1\ a x_j \rightarrow_{\texttt{min}} S_1\ a$

5 Distributed Breadth-First Search (BFS)

BFS is a fundamental technique, inherently *parallel*, or so it appears. There are a number of distributed BFS algorithms to make BFS run faster on parallel and distributed systems, such as Synchronous BFS [22], Asynchronous BFS [22], an improved Asynchronous BFS with known graph diameter [22], Layered BFS [22], Hybrid BFS [22].

Our previous research proposed a P algorithm to find disjoint paths using BFS, and empirical results show that BFS can leverage the parallel and distributed characteristics of P systems [17]. In this paper, we first present a P implementation of synchronous BFS (SyncBFS) and discuss how SyncBFS succeeds in the synchronous mode but *fails* in the asynchronous mode. Next, we propose a P implementation of an algorithm which works correctly in the asynchronous mode, the simple Asynchronous BFS (AsyncBFS) algorithm, and we show how it works in both synchronous and asynchronous scenarios.

5.1 Synchronous BFS

Initially, the source cell broadcasts out a search token. On receiving the search token, an unmarked cell marks itself and chooses one of the cells from which the search token arrived as its parent. Then in the first round after the cell gets marked, it broadcasts a search token to all its neighbors [14]. SyncBFS is a "wave" algorithm and it produces a BFS spanning tree in synchronous mode, as shown in Figure 5. However, it often fails in asynchronous mode, as shown in Figure 6.

Algorithm 8 (Synchronous BFS)

Input: Same as in Algorithm 2.

Synchronous output: All cells end in the same final state, S_5. On completion, each cell, σ_i, still contains its cell ID object, ι_i. The source cell, σ_s, is still decorated with one a. All other cells contain BFS spanning tree pointer objects, indicating predecessors, p_j. Also, cells may contain "garbage" objects, which can be cleared, by using a few more steps.

Table 8 shows the BFS spanning tree built by this algorithm (in the synchronous mode), for the P system of Figure 5 (there is only one BFS tree in this case).

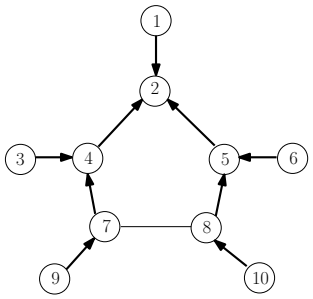


Fig. 5. BFS spanning tree.

Table 8. Partial Trace of Algorithm 8 for Figure 5 in synchronous mode.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5
0	$S_3\ \iota_1 n_2$	$S_3\ \iota_2 n_1 n_4 n_5$	$S_3\ \iota_3 n_4$	$S_3\ \iota_4 n_2 n_3 n_7$	$S_3\ \iota_5 n_2 n_6 n_8$
8	$S_5\ \iota_1 p_2 \ldots$	$S_5\ \iota_2 a \ldots$	$S_5\ \iota_3 p_4 \ldots$	$S_5\ \iota_4 p_2 \ldots$	$S_5\ \iota_5 p_2 \ldots$
Step#	σ_6	σ_7	σ_8	σ_9	σ_{10}
0	$S_3\ \iota_6 n_5$	$S_3\ \iota_7 n_4 n_8 n_9$	$S_3\ \iota_8 n_5 n_7 n_{10}$	$S_3\ \iota_9 n_7$	$S_3\ \iota_{10} n_8$
8	$S_5\ \iota_6 p_5 \ldots$	$S_5\ \iota_7 p_4 \ldots$	$S_5\ \iota_8 p_5 \ldots$	$S_5\ \iota_9 p_7 \ldots$	$S_5\ \iota_{10} p_8 \ldots$

3. Rules in state S_3:
   1 $S_3\ a \rightarrow_{\texttt{min}} S_4\ a$
   2 $S_3\ c_j n_j \rightarrow_{\texttt{min}} S_4\ p_j$
4. Rules for state S_4:
   1 $S_4\ n_j \rightarrow_{\texttt{min}} S_5\ (c_i) \downarrow_j |_{\iota_i}$
   2 $S_4 \rightarrow_{\texttt{min}} S_5$
5. Rules for state S_5:
   1 $S_5\ c_j \rightarrow_{\texttt{min}} S_5$

However, if Algorithm 8 runs in asynchronous mode, the result is still a *spanning tree*, but *not* necessarily a *BFS spanning tree*, as illustrated in Table 9 and Figure 6. The search token from cell σ_2 to σ_5 is delayed and arrives in cell σ_5 after σ_5 records its parent as σ_8. The resulting spanning tree is not a BFS spanning tree.

Table 9. Partial Trace of Algorithm 8 for Figure 6 in asynchronous mode.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5
0	$S_3\ \iota_1 n_2$	$S_3\ \iota_2 n_1 n_4 n_5$	$S_3\ \iota_3 n_4$	$S_3\ \iota_4 n_2 n_3 n_7$	$S_3\ \iota_5 n_2 n_6 n_8$
14	$S_5\ \iota_1 p_1 \ldots$	$S_5\ \iota_2 a \ldots$	$S_5\ \iota_3 p_4 \ldots$	$S_5\ \iota_4 p_2 \ldots$	$S_5\ \iota_5 p_8 \ldots$
Step#	σ_6	σ_7	σ_8	σ_9	σ_{10}
0	$S_3\ \iota_6 n_5$	$S_3\ \iota_7 n_4 n_8 n_9$	$S_3\ \iota_8 n_5 n_7 n_{10}$	$S_3\ \iota_9 n_7$	$S_3\ \iota_{10} n_8$
14	$S_5\ \iota_6 p_5 \ldots$	$S_5\ \iota_7 p_4 \ldots$	$S_5\ \iota_8 p_7 \ldots$	$S_5\ \iota_9 p_7 \ldots$	$S_5\ \iota_{10} p_8 \ldots$

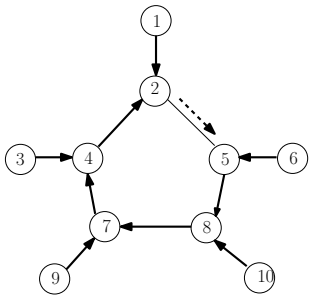


Fig. 6. BFS spanning tree output of Algorithm 8 in an asynchronous scenario.

5.2 Asynchronous BFS

Asynchronous BFS (AsyncBFS) algorithm is not just a asynchronous version of SyncBFS [14], as previously discussed in the asynchronous mode of SyncBFS. It has modifications to correct the parent destination, therefore obtaining a BFS spanning tree.

Although the known problem of AsyncBFS is that there is no way to know when there are no further parent corrections to make, i.e. it never produces the tree structure output. However, in P systems, there is no such problem, because the objects in cells are actually the tree link output. Thus, P systems provides a favorable way to implement this algorithm, which does not require other augmenting approaches, such as adding acknowledgments, convergecasting acknowledgments, bookkeeping, etc [14].

Algorithm 9 (Asynchronous BFS)

Input: Same as in Algorithms 2 (and 8).

Output: Similar to Algorithm 8 (running in synchronous mode), but the final state is S_4. Also, cells may contain "garbage" objects, which can be cleared, by using a few more steps.

Table 10 shows the BFS spanning tree built by this algorithm, for the P system of Figure 5 (there is only one BFS tree in this case).

Table 10. Partial Trace of Algorithm 9 for Figure 5.

Step#	σ_1	σ_2	σ_3	σ_4	σ_5
0	$S_3\ \iota_1 n_2$	$S_3\ \iota_2 n_1 n_4 n_5$	$S_3\ \iota_3 n_4$	$S_3\ \iota_4 n_2 n_3 n_7$	$S_3\ \iota_5 n_2 n_6 n_8$
5	$S_4 p_2 \ldots$	$S_4 a \ldots$	$S_4 p_4 \ldots$	$S_4 p_2 \ldots$	$S_4 p_2 \ldots$
Step#	σ_6	σ_7	σ_8	σ_9	σ_{10}
0	$S_3\ \iota_6 n_5$	$S_3\ \iota_7 n_4 n_8 n_9$	$S_3\ \iota_8 n_5 n_7 n_{10}$	$S_3\ \iota_9 n_7$	$S_3\ \iota_{10} n_8$
5	$S_4\ \iota_6 p_5 \ldots$	$S_4\ \iota_7 p_4 \ldots$	$S_4\ \iota_8 p_5 \ldots$	$S_4\ \iota_9 p_7 \ldots$	$S_4\ \iota_{10} p_8 \ldots$

3. Rules in state S_3:
   1 $S_3 \ \rightarrow_{\texttt{min}} S_4 \ h|_a$
   2 $S_3 \ n_j \rightarrow_{\texttt{min}}$
   $S_4 \ m_j \ (c_i t g g u u) \downarrow_j |_{a\iota_i}$
   3 $S_3 \ c_j n_j \rightarrow_{\texttt{min}} S_4 \ p_j m_j|_t$
   4 $S_3 \ \rightarrow_{\texttt{min}} S_4 \ (c_i t) \downarrow_j |_{t n_j \iota_i}$
   5 $S_3 \ g u \rightarrow_{\texttt{min}} S_4 \ h \ (g g u u) \updownarrow |_t$
   6 $S_3 \ g u \rightarrow_{\texttt{max}} S_4 \ h \ (g u) \updownarrow |_t$
   7 $S_3 \ n_j \rightarrow_{\texttt{min}} S_4 \ m_j|_t$
   8 $S_3 \ t \rightarrow_{\texttt{max}} S_4$
4. Rules for state S_4:
   1 $S_4 \ g h \rightarrow_{\texttt{max}} S_4 \ |_t$
   2 $S_4 \ p_j \rightarrow_{\texttt{min}} S_4 \ |_{ht}$
   3 $S_4 \ c_j m_j \rightarrow_{\texttt{min}} S_4 \ p_j|_{ht}$
   4 $S_4 \ c_j \rightarrow_{\texttt{min}} S_4 \ |_t$
   5 $S_4 \ m_j \rightarrow_{\texttt{min}} S_4 \ (c_i t) \downarrow_j |_{ht\iota_i}$
   6 $S_4 \ u \rightarrow_{\texttt{min}} S_4 \ h \ (g g u u) \updownarrow |_{ht}$
   7 $S_4 \ u \rightarrow_{\texttt{max}} S_4 \ h \ (g u) \updownarrow |_{ht}$
   8 $S_4 \ h \rightarrow_{\texttt{max}} S_4 \ |_t$
   9 $S_4 \ g u \rightarrow_{\texttt{max}} S_4$
   10 $S_4 \ g u \rightarrow_{\texttt{max}} S_4|_t$
   11 $S_4 \ t \rightarrow_{\texttt{max}} S_4$

5.3 Echo Algorithm

The Echo algorithm shares the similar "wave" characteristics of distributed BFS algorithms, but, as discussed in Section 3, it only builds a spanning tree, not necessarily a BFS spanning tree.

Algorithm 10 (Echo Algorithm)

Input: Same as in Algorithms 2 (and 8).

Output: All cells end in the same final state, S_4. On completion, each cell, σ_i, still contains its cell ID object, ι_i. he source cell, σ_s, is still decorated with an object, a. All other cells contain a spanning tree pointer objects, indicating predecessors, p_j.

Table 11 and 12 show two spanning trees, built by this algorithm, for the P system of Figures 1 and 2, in synchronous and asynchronous modes, respectively.

Table 11. Partial Trace of Algorithm 10 for Figure 1 in synchronous mode.

Step#	σ_1	σ_2	σ_3	σ_4
0	$S_3 \ \iota_1 a n_2 n_3 n_4$	$S_3 \ \iota_2 n_1 n_3 n_4$	$S_3 \ \iota_3 n_1 n_2 n_4$	$S_3 \ \iota_4 n_1 n_2 n_3$
4	$S_4 \ \iota_1 a$	$S_4 \ \iota_2 p_1$	$S_4 \ \iota_3 p_1$	$S_4 \ \iota_4 p_1$

Table 12. Partial Trace of Algorithm 10 for Figure 2 in asynchronous mode.

Step#	σ_1	σ_2	σ_3	σ_4
0	$S_3 \ \iota_1 a n_2 n_3 n_4$	$S_3 \ \iota_2 n_1 n_3 n_4$	$S_3 \ \iota_3 n_1 n_2 n_4$	$S_3 \ \iota_4 n_1 n_2 n_3$
4	$S_4 \ \iota_1 a$	$S_4 \ \iota_2 p_1$	$S_4 \ \iota_3 p_2$	$S_4 \ \iota_4 p_3$

3. Rules in state S_3:
   1 $S_3\ n_j \rightarrow_{\texttt{min}} S_4\ w_j\ (c_i t) \downarrow_j |_{a\iota_i}$
   2 $S_3\ c_j n_j n_k \rightarrow_{\texttt{min}} S_4\ p_j w_k\ (c_i t) \downarrow_k |_{\iota_i}$
   3 $S_3\ c_j n_j \rightarrow_{\texttt{min}} S_4\ p_j\ (c_i t) \downarrow_j |_{\iota_i}$
   4 $S_3\ n_j \rightarrow_{\texttt{min}} S_4\ w_j\ (c_i t) \downarrow_j |_{t\iota_i}$
   5 $S_3\ t \rightarrow_{\texttt{max}} S_4$
4. Rules for state S_4:
   1 $S_4\ w_j \rightarrow_{\texttt{min}} S_4\ |_{c_j}$
   2 $S_4\ w_j p_k \rightarrow_{\texttt{min}} S_4\ w_j p_k$
   3 $S_4\ w_j a \rightarrow_{\texttt{min}} S_4\ w_j a$
   4 $S_4\ c_j \rightarrow_{\texttt{min}} S_4$
   5 $S_4\ p_j \rightarrow_{\texttt{min}} S_4\ p_j\ (c_i t) \downarrow_j |_{t\iota_i}$
   6 $S_4\ t \rightarrow_{\texttt{max}} S_4$

6 Complexity

All our distributed DFS and BFS implementations, except the SoD implementation, assume that each cells knows the IDs of its neighbors (parents and children). Our SoD implementation assumes that each cell knows the labels of its adjacent arcs (incoming and outgoing). In the complexity analysis, we skip over a preliminary phase which could build such knowledge, see Algorithm 1.

All our P system DFS implementations take one final step, to prompt the source cell to discard the token; we also omit this step in the complexity analysis. Moreover, there is one beginning step in our implementations for Awerbuch (rule 3.1) and Cidon (rules 3.1, 3.2), which instantiates initial list objects. These steps can be included in Algorithm 1. However, we do not follow this approach, because we want to keep Algorithm 1 a common preliminary phase for all our algorithms. We also skip these beginning steps, in the complexity analysis.

Table 13 shows the resulting complexity of our P system DFS implementations, in terms of P steps. The runtime complexity of our P system implementations is exactly the same as for the standard distributed DFS algorithms. The complexity of our SOD algorithm must be considered with a big grain of salt, for the reasons explained in the description of Algorithm 7 (high-level pseudo-code).

Table 13. DFS algorithms comparisons and complexity (P steps) of Figure 3.

Algorithm	P Steps	Time units	Messages	Notes
Classical	18	$2M$	$2M$	Local cell IDs
Awerbuch	22	$4N-2$	$4M$	Local cell IDs
Cidon	10	$2N-2$	$\leq 4M$	Local cell IDs
Sharma	10	$2N-2$	$\leq 2N-2$	Global cell IDs
SOD	10?	$2N-2$	$\leq 2N-2$	Sense of Direction (Z_n)
Makki	9	$(1+r)N$	$(1+r)N$	Global cell IDs (or SOD)

Table 14 shows the runtime complexity of our P system SyncBFS and AsyncBFS implementations, which is consistent with the runtime complexity of the standard algorithms.

Table 14. BFS algorithms comparisons and complexity (P steps) of Figure 5.

Algorithm	P Steps	Time units	Messages	Notes
Sync	8	$O(D)$	$O(M)$	Local IDs
Simple Async	5	$O(DN)$	$O(NM)$	Local IDs
Simple Async2	?	$O(D^2)$	$O(DM)$	D and Local IDs
Layered Async	?	$O(D^2)$	$O(M + DN)$	Local IDs
Hybrid Async	?	$O(Dk + D^2/k)$	$O(Mk + DN/k)$	Local IDs

7 Conclusions

We proposed a new approach to fully asynchronous P systems, and a matching complexity measure, both inspired from the field of distributed algorithms. We validated our approach by implementing several well-known distributed depth-first search (DFS) and breadth-first search (BFS) algorithms. We believe that these are the first P implementations of the standard distributed DFS and BFS algorithms. Empirical results show that, in terms of P steps, the runtime complexity of our distributed P algorithms is the same as the runtime complexity of standard distributed DFS and BFS.

Several interesting questions remain open. We intend to complete this quest by completing the implementation of the SOD algorithm and by implementing three other, more sophisticated, distributed BFS algorithms and compare their performance against the standard versions. We also intend to elaborate the foundations of fully asynchronous P systems and further validate this, by investigating a few famous critical problems, such as building minimal spanning trees. Finally, we intend to formulate fundamental distributed asynchronous concepts, such as fairness, safety and liveness, and investigate methods for their proofs.

References

1. Awerbuch, B.: A new distributed depth-first-search algorithm. Information Processing Letters 20(3), 147 – 150 (1985), http://www.sciencedirect.com/science/article/B6V0F-482R9G2-S/2/22537b651ddd5c1a0e3ae5d5ba723079
2. Bernardini, F., Gheorghe, M., Margenstern, M., Verlan, S.: How to synchronize the activity of all components of a P system? Int. J. Found. Comput. Sci. 19(5), 1183–1198 (2008)
3. Casiraghi, G., Ferretti, C., Gallini, A., Mauri, G.: A membrane computing system mapped on an asynchronous, distributed computational environment. In: Freund, R., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 3850, pp. 159–164. Springer (2005)
4. Cavaliere, M., Egecioglu, O., Ibarra, O., Ionescu, M., Pun, G., Woodworth, S.: Asynchronous spiking neural p systems: Decidability and undecidability. In: Garzon, M., Yan, H. (eds.) DNA Computing, Lecture Notes in Computer Science, vol. 4848, pp. 246–255. Springer Berlin / Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-77962-9_26

5. Cavaliere, M., Ibarra, O.H., Pun, G., Egecioglu, O., Ionescu, M., Woodworth, S.: Asynchronous spiking neural p systems. Theor. Comput. Sci. 410, 2352–2364 (May 2009), http://portal.acm.org/citation.cfm?id=1539070.1540146
6. Cavaliere, M., Sburlan, D.: Time and synchronization in membrane systems. Fundam. Inf. 64, 65–77 (July 2004), http://portal.acm.org/citation.cfm?id=1227085.1227092
7. Cidon, I.: Yet another distributed depth-first-search algorithm. Inf. Process. Lett. 26, 301–305 (1988)
8. Dinneen, M.J., Kim, Y.B., Nicolescu, R.: Edge- and node-disjoint paths in P systems. Electronic Proceedings in Theoretical Computer Science 40, 121–141 (2010)
9. Dinneen, M.J., Kim, Y.B., Nicolescu, R.: Edge- and vertex-disjoint paths in P modules. In: Ciobanu, G., Koutny, M. (eds.) Workshop on Membrane Computing and Biologically Inspired Process Calculi. pp. 117–136 (2010)
10. Dinneen, M.J., Kim, Y.B., Nicolescu, R.: P systems and the Byzantine agreement. Journal of Logic and Algebraic Programming 79(6), 334–349 (2010), http://www.sciencedirect.com/science/article/B6W8D-4YPPPW1-2/2/17b82b2cdd8f159b7fea380939193e4d
11. Freund, R.: Asynchronous p systems and p systems working in the sequential mode. In: Mauri, G., Paun, G., Prez-Jimnez, M., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 3365, pp. 36–62. Springer Berlin / Heidelberg (2005), http://dx.doi.org/10.1007/978-3-540-31837-8_3
12. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Depth-first search with p systems. In: Proceedings of the 11th international conference on Membrane computing. pp. 257–264. CMC'10, Springer-Verlag, Berlin, Heidelberg (2010), http://portal.acm.org/citation.cfm?id=1946067.1946090
13. Kleijn, J., Koutny, M.: Synchrony and asynchrony in membrane systems. In: Membrane Computing, WMC2006, Leiden, Revised, Selected and Invited Papers, LNCS 4361. pp. 66–85. Springer (2006)
14. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
15. Makki, S.A.M., Havas, G.: Distributed algorithms for depth-first search. Inf. Process. Lett. 60, 7–12 (October 1996), http://portal.acm.org/citation.cfm?id=244081.244085
16. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Discovering the membrane topology of hyperdag P systems. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5957, pp. 410–435. Springer-Verlag (2009)
17. Nicolescu, R., Wu, H.: BFS solution for disjoint paths in P systems. Report CDMTCS-399, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand (March 2011), http://www.cs.auckland.ac.nz/CDMTCS//researchreports/399radu.pdf
18. Pan, L., Zeng, X., Zhang, X.: Time-free spiking neural p systems. Neural Computation 0(0), 1–23 (2011), http://www.mitpressjournals.org/doi/abs/10.1162/NECO_a_00115
19. Păun, G.: Introduction to membrane computing. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.) Applications of Membrane Computing, pp. 1–42. Natural Computing Series, Springer-Verlag (2006)
20. Păun, G., Centre, T., Science, C.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (1998)

21. Sharma, M.B., Iyengar, S.S.: An efficient distributed depth-first-search algorithm. Inf. Process. Lett. 32, 183–186 (September 1989), http://portal.acm.org/citation.cfm?id=69686.69691
22. Tel, G.: Introduction to Distributed Algorithms. Cambridge University Press (2000)
23. Tsin, Y.H.: Some remarks on distributed depth-first search. Inf. Process. Lett. 82, 173–178 (May 2002), http://portal.acm.org/citation.cfm?id=585580.585581
24. Yuan, Z., Zhang, Z.: Asynchronous spiking neural p system with promoters. In: Proceedings of the 7th international conference on Advanced parallel processing technologies. pp. 693–702. APPT'07, Springer-Verlag, Berlin, Heidelberg (2007), http://portal.acm.org/citation.cfm?id=1785246.1785331

A Appendix

Table 15. Awerbuch DFS algorithm traces (steps 0, . . . , 15) of Figure 3 in synchronous mode, where σ_1 is the source cell.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
0	$S_0 a_1$	$S_0 \iota_2$	$S_0 \iota_3$	$S_0 \iota_4$	$S_0 \iota_5$	$S_0 \iota_6$
1	$S_1 a_1 y$	$S_0 \iota_2 z$	$S_0 \iota_3 z$	$S_0 \iota_4 z$	$S_0 \iota_5 z$	$S_0 \iota_6$
2	$S_2 a_1 z^2$	$S_1 \iota_2 n_1 yz$	$S_0 \iota_3 z^2$	$S_1 \iota_4 n_1 yz$	$S_0 \iota_5 z$	$S_0 \iota_6$
3	$S_3 a_1 n_2 n_4$	$S_2 \iota_2 n_1 n_4 z$	$S_1 \iota_3 n_2 n_4 yz$	$S_2 \iota_4 n_1 n_2 z^2$	$S_1 \iota_5 n_4 yz$	$S_0 \iota_6 z^2$
4	$S_4 a_1 m_2 m_4 n_2 n_4$	$S_3 \iota_2 n_1 n_3 n_4$	$S_2 \iota_3 n_2 n_4 n_5 z$	$S_3 \iota_4 n_1 n_2 n_3 n_5$	$S_2 \iota_5 n_3 n_4 z$	$S_1 \iota_6 n_3 n_5 y$
5	$S_5 a_1 m_2 m_4 n_2 n_4$	$S_4 \iota_2 m_1 m_3 m_4 n_1 n_3 n_4$	$S_3 \iota_3 n_2 n_4 n_5 n_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 v_1$	$S_3 \iota_5 n_3 n_4 n_6$	$S_2 \iota_6 n_3 n_5$
6	$S_6 a b_2 b_4 \iota_1 m_2 m_4 n_2$ $n_4 w_2 w_4$	$S_4 \iota_2 m_1 m_3 m_4 n_1 n_3 n_4$ u_1	$S_4 \iota_3 m_2 m_4 m_5 m_6 n_2 n_4$ $n_5 n_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$	$S_3 \iota_6 n_3 n_5$
7	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_4 c_1 \iota_2 m_1 m_3 m_4 n_1 n_3$ $n_4 t u_1$	$S_4 \iota_3 m_2 m_4 m_5 m_6 n_2 n_4$ $n_5 n_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$	$S_4 \iota_6 m_3 m_5 n_3 n_5$
8	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_5 \iota_2 m_3 m_4 n_3 n_4 p_1 t\ u_1$	$S_4 \iota_3 m_2 m_4 m_5 m_6 n_2 n_4$ $n_5 n_6 v_2$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 v_2$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$	$S_4 \iota_6 m_3 m_5 n_3 n_5$
9	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_6 b_3 b_4 \iota_2 m_3 m_4 n_3 n_4$ $p_1 t u_1 w_3 w_4$	$S_4 \iota_3 m_2 m_4 m_5 m_6 n_2 n_4$ $n_5 n_6 u_2$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$	$S_4 \iota_6 m_3 m_5 n_3 n_5$
10	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_4 c_2 \iota_3 m_2 m_4 m_5 m_6 n_2$ $n_4 n_5 n_6 t u_2$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$	$S_4 \iota_6 m_3 m_5 n_3 n_5$
11	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_5 \iota_3 m_4 m_5 m_6 n_4 n_5 n_6$ $p_2 t u_2$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2 v_3$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$ v_3	$S_4 \iota_6 m_3 m_5 n_3 n_5 v_3$
12	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_6 b_4 b_5 b_6 \iota_3 m_4 m_5 n_6$ $n_4 n_5 n_6 p_2 t u_2 w_4 w_5\ w_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2 u_3$	$S_4 \iota_5 m_3 m_4 m_6 n_3 n_4 n_6$ u_3	$S_4 \iota_6 m_3 m_5 n_3 n_5 u_3$
13	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 m_6 n_4 n_5 n_6 p_2$ $u_2 u_5$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2 u_3$	$S_4 c_3 \iota_5 m_3 m_4 m_6 n_3 n_4$ $n_6 t u_3$	$S_4 \iota_6 m_3 m_5 n_3 n_5 u_3$
14	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 m_6 n_4 n_5 n_6 p_2$ $u_2 u_5$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2 u_3 v_5$	$S_5 \iota_5 m_4 m_6 n_4 n_6 p_3 t\ u_3$	$S_4 \iota_6 m_3 m_5 n_3 n_5 u_3 v_5$
15	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 m_6 n_4 n_5 n_6 p_2$ $u_2 u_5$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2$ $n_3 n_5 u_1 u_2 u_3 u_5$	$S_6 b_4 b_6 \iota_5 m_4 m_6 n_4 n_6$ $p_3 t u_3 w_4 w_6$	$S_4 \iota_6 m_3 m_5 n_3 n_5 u_3 u_5$

Table 16. Awerbuch DFS algorithm traces (steps 16, ..., 27) of Figure 3 in synchronous mode, where σ_1 is the source cell.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
16	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 m_6 n_4 n_5 n_6 p_2 u_2 u_5$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2 n_3 n_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 m_4 n_4 n_6 p_3 u_3 u_6$	$S_4 c_5 \iota_6 m_3 m_5 n_3 n_5 t u_3 u_5$
17	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 m_6 n_4 n_5 n_6 p_2 u_2 u_5 v_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2 n_3 n_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 m_4 n_4 n_6 p_3 u_3 u_6$	$S_5 \iota_6 m_3 n_3 p_5 t u_3 u_5$
18	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 m_6 n_4 n_5 n_6 p_2 u_2 u_5 u_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2 n_3 n_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 m_4 n_4 n_6 p_3 u_3 u_6$	$S_6 b_3 \iota_6 m_3 n_3 p_5 t u_3 u_5 w_3$
19	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 n_4 n_5 n_6 p_2 u_2 u_5 u_6$	$S_4 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2 n_3 n_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 m_4 n_4 n_6 p_3 t u_3 u_6 x_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
20	$S_7 a \iota_1 m_4 n_2 n_4 u_2$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3$	$S_7 \iota_3 m_4 n_4 n_5 n_6 p_2 u_2 u_5 u_6$	$S_4 c_5 \iota_4 m_1 m_2 m_3 m_5 n_1 n_2 n_3 n_5 t u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
21	$S_7 a \iota_1 m_4 n_2 n_4 u_2 v_4$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3\ v_4$	$S_7 \iota_3 m_4 n_4 n_5 n_6 p_2 u_2 u_5 u_6 v_4$	$S_5 \iota_4 m_1 m_2 m_3 n_1 n_2 n_3 p_5 t u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
22	$S_7 a \iota_1 m_4 n_2 n_4 u_2 u_4$	$S_7 \iota_2 m_4 n_3 n_4 p_1 u_1 u_3\ u_4$	$S_7 \iota_3 m_4 n_4 n_5 n_6 p_2 u_2 u_4 u_5 u_6$	$S_6 b_1 b_2 b_3 \iota_4 m_1 m_2 m_3 n_1 n_2 n_3 p_5 t u_1 u_2 u_3 u_5 w_1 w_2 w_3$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
23	$S_7 a \iota_1 n_2 n_4 u_2 u_4$	$S_7 \iota_2 n_3 n_4 p_1 u_1 u_3 u_4$	$S_7 \iota_3 n_4 n_5 n_6 p_2 u_2 u_4 u_5 u_6$	$S_7 \iota_4 n_1 n_2 n_3 p_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 t u_3 u_4\ u_6 x_4$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
24	$S_7 a \iota_1 n_2 n_4 u_2 u_4$	$S_7 \iota_2 n_3 n_4 p_1 u_1 u_3 u_4$	$S_7 \iota_3 n_4 n_5 n_6 p_2 t u_2 u_4 u_5 u_6 x_5$	$S_7 \iota_4 n_1 n_2 n_3 p_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
25	$S_7 a \iota_1 n_2 n_4 u_2 u_4$	$S_7 \iota_2 n_3 n_4 p_1 t u_1 u_3\ u_4 x_3$	$S_7 \iota_3 n_4 n_5 n_6 p_2 u_2 u_4 u_5 u_6$	$S_7 \iota_4 n_1 n_2 n_3 p_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
26	$S_7 a \iota_1 n_2 n_4 t u_2 u_4\ x_2$	$S_7 \iota_2 n_3 n_4 p_1 u_1 u_3 u_4$	$S_7 \iota_3 n_4 n_5 n_6 p_2 u_2 u_4 u_5 u_6$	$S_7 \iota_4 n_1 n_2 n_3 p_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$
27	$S_7 a \iota_1 n_2 n_4 u_2 u_4$	$S_7 \iota_2 n_3 n_4 p_1 u_1 u_3 u_4$	$S_7 \iota_3 n_4 n_5 n_6 p_2 u_2 u_4 u_5 u_6$	$S_7 \iota_4 n_1 n_2 n_3 p_5 u_1 u_2 u_3 u_5$	$S_7 \iota_5 n_4 n_6 p_3 u_3 u_4 u_6$	$S_7 \iota_6 n_3 p_5 u_3 u_5$

Simulating Spiking Neural P Systems Without Delays Using GPUs

Francis Cabarle[1], Henry 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: 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. We present in this paper our work regarding simulating a type of P system known as a spiking neural P system (SNP system) using graphics processing units (GPUs). GPUs, because of their architectural optimization for parallel computations, are well-suited for highly parallelizable problems. Due to the advent of general purpose GPU computing in recent years, GPUs are not limited to graphics and video processing alone, but include computationally intensive scientific and mathematical applications as well. Moreover P systems, including SNP systems, are inherently and maximally parallel computing models whose inspirations are taken from the functioning and dynamics of a living cell. In particular, SNP systems try to give a modest but formal representation of a special type of cell known as the neuron and their interactions with one another. The nature of SNP systems allowed their representation as matrices, which is a crucial step in simulating them on highly parallel devices such as GPUs. The highly parallel nature of SNP systems necessitate the use of hardware intended for parallel computations. The simulation algorithms, design considerations, and implementation are presented. Finally, simulation results, observations, and analyses using an SNP system that generates all numbers in $\mathbb{N}$ - {1} are discussed, as well as recommendations for future work.

Key words: Membrane computing, Parallel computing, GPU computing

1 Introduction

1.1 Parallel computing: Via graphics processing units (GPUs)

The trend for massively parallel computation is moving from the more common multi-core CPUs towards GPUs for several significant reasons [13, 14]. One impor-

tant reason for such a trend in recent years include the low consumption in terms of power of GPUs compared to setting up machines and infrastructure which will utilize multiple CPUs in order to obtain the same level of parallelization and performance [15]. Another more important reason is that GPUs are architectured for *massively parallel computations* since unlike most general purpose multicore CPUs, a large part of the architecture of GPUs are devoted to parallel execution of arithmetic operations, and not on control and caching just like in CPUs [13, 14]. Arithmetic operations are at the heart of many basic operations as well as scientific computations, and these are performed with larger speedups when done in parallel as compared to performing them sequentially. In order to perform these arithmetic operations on the GPU, there is a set of techniques called *GPGPU* (General Purpose computations on the GPU) coined by Mark Harris in 2002 which allows programmers to do computations on GPUs and not be limited to just graphics and video processing alone [1].

1.2 Parallel computing: Via Membranes

Membrane computing or its more specific counterpart, a *P system*, is a Turing complete computing model (for several P system variants) that perform computations nondeterministically, exhausting all possible computations at any given time. This type of unconventional model of computation was introduced by Gheorghe Păun in 1998 and takes inspiration and abstraction, similar to other members of *Natural computing* (e.g. DNA/molecular computing, neural networks, quantum computing), from nature [6, 7]. Specifically, P systems try to mimic the constitution and dynamics of the living cell: the multitude of elements inside it, and their interactions within themselves and their environment, or outside the cell's *skin* (the cell's outermost membrane). Before proceeding, it is important to clarify what is meant when it is said that nature *computes*, particularly life or the cell: computation in this case involves reading information from memory from past or present stimuli, rewrite and retrieve this data as a stimuli from the environment, process the gathered data and act accordingly due to this processing [2]. Thus, we try to extend the classical meaning of computation presented by Allan Turing.

SN P systems differ from other types of P systems precisely because they are $mono - membranar$ and the working alphabet contains only *one object type*. These characteristics, among others, are meant to capture the workings of a special type of cell known as the *neuron*. Neurons, such as those in the human brain, communicate or 'compute' by sending indistinct signals more commonly known as action potential or *spikes* [3]. $Information$ is then communicated and encoded not by the spikes themselves, since the spikes are unrecognizable from one another, but by (a) the time elapsed between spikes, as well as (b) the number of spikes sent/received from one neuron to another, oftentimes under a certain time interval [3].

It has been shown that SN P systems, given their nature, are representable by matrices [4, 5]. This representation allows design and implementation of an SN P system simulator using parallel devices such as GPUs.

1.3 Simulating SNP systems in GPUs

Since the time P systems were presented, many simulators and software applications have been produced [10]. In terms of *High Performance Computing*, many P system simulators have been also designed for clusters of computers [11], for reconfigurable hardware as in FPGAs [12], and even for GPUs [9, 8]. All of these efforts have shown that parallel architectures are well-suited in performance to simulate P systems. However, these previous works on hardware are designed to simulate *cell-like* P system variants, which are among the first P system variants to have been introduced. Thus, the efficient simulation of SNP systems is a new challenge that requires novel attempts.

A matrix representation of SN P systems is quite intuitive and natural due to their graph-like configurations and properties (as will be further shown in the succeeding sections such as in subsection 2.1).

On the other hand, *linear algebra* operations have been efficiently implemented on parallel platforms and devices in the past years. For instance, there is a large number of algorithms implementing $matrix - matrix$ and $vector - matrix$ operations on the GPU. These algorithms offer huge performance since dense linear algebra readily maps to the data-parallel architecture of GPUs [16, 17].

It would thus seem then that a matrix represented SN P system simulator implementation on highly parallel computing devices such as GPUs be a natural confluence of the earlier points made. The matrix representation of SN P systems bridges the gap between the theoretical yet still computationally powerful SN P systems and the applicative and more tangible GPUs, via an SN P system simulator.

The design and implementation of the simulator, including the algorithms deviced, architectural considerations, are then implemented using CUDA. The Compute Unified Device Architecture (CUDA) programming model, launched by NVIDIA in mid-2007, is a hardware and software architecture for issuing and managing computations on their most recent GPU families (G80 family onward), making the GPU operate as a highly parallel computing device [15]. CUDA programming model extends the widely known ANSI C programming language (among other languages which can interface with CUDA), allowing programmers to easily design the code to be executed on the GPU, avoiding the use of low-level graphical primitives. CUDA also provides other benefits for the programmer such as abstracted and automated scaling of the parallel executed code.

This paper starts out by introducing and defining the type of SNP system that will be simulated. Afterwards the NVIDIA CUDA model and architecture are discussed, baring the scalability and parallelization CUDA offers. Next, the design of the simulator, constraints and considerations, as well as the details of the algorithms used to realize the SNP system are discussed. The simulation results are presented next, as well as observations and analysis of these results. The paper ends by providing the conclusions and future work.

The objective of this work is to continue the creation of P system simulators , in this particular case an SN P system, using highly parallel devices such as GPUs.

Fidelity to the computing model (the type of SNP system in this paper) is a part of this objective.

2 Spiking neural p systems

2.1 Computing with SN P systems

The type of SNP systems focused on by this paper (scope) are those without delays i.e. those that spike or transmit signals the moment they are able to do so [4, 5]. Variants which allow for delays before a neuron produces a spike, are also available [3]. An SNP system without delay is of the form:

Definition 1.

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

where:

1. $O = \{a\}$ *is the alphabet made up of only one object, the system spike* a.
2. $\sigma_1, \ldots, \sigma_m$ *are* m *number of neurons of the form*

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:

a) $n_i \geq 0$ *gives the initial number of as i.e. spikes contained in neuron* σ_i

b) R_i *is a finite set of rules of with two forms:*

(b-1) $E/a^c \rightarrow a^p$, *are known as Spiking rules, where* E *is a regular expression over* a, *and* $c \geq 1$, *such that* $p \geq 1$ *number of spikes are produced, one for each adjacent neuron with* σ_i *as the originating neuron and* $a^c \in L(E)$.

(b-2) $a^s \rightarrow \lambda$, *are known as Forgetting rules, for* $s \geq 1$, *such that for each rule* $E/a^c \rightarrow a$ *of type (b-1) from* R_i, $a^s \notin L(E)$.

(b-3) $a^k \rightarrow a$, *a special case of (b-1) where* $E = a^c$, $k \geq c$.

3. $syn = \{(i, j) \,|\, 1 \leq i, j \leq m, i \neq j\}$ *are the synapses i.e. connection between neurons.*
4. $in, out \in \{1, 2, \ldots, m\}$ *are the input and output neurons, respectively.*

Furthermore, rules of type (b-1) are applied if σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$. Using this type of rule uses up or consumes k spikes from the neuron, producing a spike to each of the neurons connected to it via a forward pointing arrow i.e. away from the neuron. In this manner, for rules of type (b-2) if σ_i contains s spikes, then s spikes are forgotten or removed once the rule is used.

The non-determinism of SN P systems comes with the fact that more than one rule of the several types are applicable at a given time, given enough spikes. The rule to be used is chosen non-deterministically in the neuron. However, only one rule can be applied or used at a given time [3, 4, 5]. The neurons in an SN P system operate in parallel and in unison, under a global clock [3]. For Figure 1

no input neuron is present, but neuron 3 is the output neuron, hence the arrow pointing towards the environment, outside the SNP system.

The SN P system in Figure 1 is Π, a 3 neuron system whose neurons are labeled (neuron $1/\sigma_1$ to neuron $3/\sigma_3$) and whose rules have a total system ordering from (1) to (5). Neuron $1/\sigma_1$ can be seen to have an initial number of spikes equal to 2 (hence the a^2 seen inside it). There is *no* input neuron, but σ_3 is the output neuron, as seen by the arrow pointing towards the environment (not to another neuron). More formally, Π can be represented as follows:

$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn, out)$ where $\sigma_1 = (2, R_1)$, $n_1 = 2$, $R_1 = \{a^2/a \rightarrow a\}$, (neurons 2 to 3 and their n_is and R_is can be similarly shown), $syn = \{(1,2),(1,3),(2,1),(2,3)\}$ are the synapses for Π, $out = \sigma_3$. This SN P system generates all numbers in the set $\mathbb{N}$ - $\{1\}$, hence it doesn't *halt*, which can be easily verified by applying the rules in Π, and checking the spikes produced by the output neuron σ_3. This generated set is the result of the computation in Π.

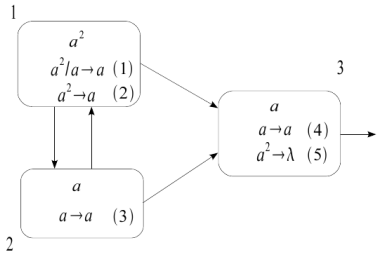


Fig. 1. An SNP P system Π, generating all numbers in $\mathbb{N}$ - $\{1\}$, from [5].

2.2 Matrix representation of SNP systems

A matrix representation of an SN P system makes use of the following vectors and matrix definitions [4, 5] . It is important to note that, just as in Figure 1, a total ordering of rules is considered.

Configuration vector C_k is the vector containing all spikes in every neuron on the *kth* computation step/time, where C_0 is the initial vector containing all spikes in the system at the beginning of the computation. For Π (in Figure 1) the initial configuration vector is $C_0 =< 2, 1, 1 >$.

Spiking vector shows at a given configuration C_k, if a rule is applicable (has value *1*) or not (has value *0* instead). For Π we have the spiking vector $S_k =< 1, 0, 1, 1, 0 >$ given C_0. Note that a 2nd spiking vector, $S_k =< 0, 1, 1, 1, 0 >$, is possible if we use rule (2) over rule (1) instead (but not both at the same time, hence we cannot have a vector equal to $< 1, 1, 1, 1, 0 >$, so this S_k is invalid). *Validity* in this case means that only one among several applicable rules is used and thus represented in the spiking vector. We can have all the possible vectors

composed of 0s and 1s with length equal to the number of rules, but have only some of them be valid, given by Ψ later at subsection 4.2.

Spiking transition matrix M_Π is a matrix comprised of a_{ij} elements where a_{ij} is given as

Definition 2.

$$a_{ij} = \begin{cases} -c, \textit{ rule } r_i \textit{ is in } \sigma_j \textit{ and is applied consuming } c \textit{ spikes;} \\ p, \textit{ rule } r_i \textit{ is in } \sigma_s \textit{ } (s \neq j \textit{ and } (s,j) \in syn) \\ \quad \textit{and is applied producing } p \textit{ spikes in total;} \\ 0, \textit{ rule } r_i \textit{ is in } \sigma_s \textit{ } (s \neq j \textit{ and } (s,j) \notin syn). \end{cases}$$

For Π, the M_Π is as follows:

$$M_\Pi = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} \tag{1}$$

In such a scheme, rows represent rules and columns represent neurons.

Finally, the following equation provides the configuration vector at the $(k+1)th$ step, given the configuration vector and spiking vector at the kth step, and M_Π:

$$C_{k+1} = C_k \ + \ S_k \cdot M_\Pi. \tag{2}$$

3 The NVIDIA CUDA architecture

NVIDIA, a well known manufacturer of GPUs, released in 2007 the CUDA programming model and architecture [15]. Using extensions of the widely known C language, a programmer can write parallel code which will then execute in multiple threads within multiple thread blocks, each contained within a grid of (thread) blocks. These grids belong to a single device i.e. a single GPU. Each device/ GPU has multiple cores, each capable of running its own *block of threads* The program run in the CUDA model scales up or down, depending on the number of cores the programmer currently has in a device. This scaling is done in a manner that is abstracted from the user, and is efficiently handled by the architecture as well. Automatic and efficient scaling is shown in Figure 2. Parallelized code will run faster with more cores than with fewer ones [14].

Figure 3 shows another important feature of the CUDA model: the host and the device parts. The host controls the execution flow while the device is a highly-parallel co-processor. Device pertains to the GPU/s of the system, while the host pertains to the CPU/s. A function known as a *kernel function*, is a function called from the host but executed in the device.

A general model for creating a CUDA enabled program is shown in Listing 1.

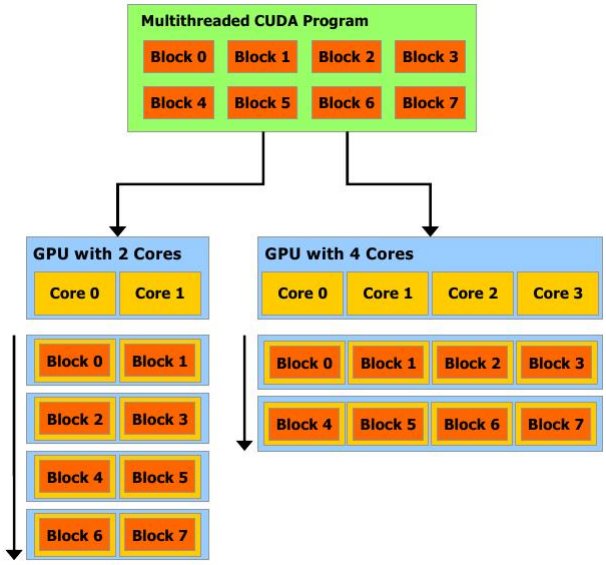


Fig. 2. NVIDIA CUDA automatic scaling, hence more cores result to faster execution, from [14].

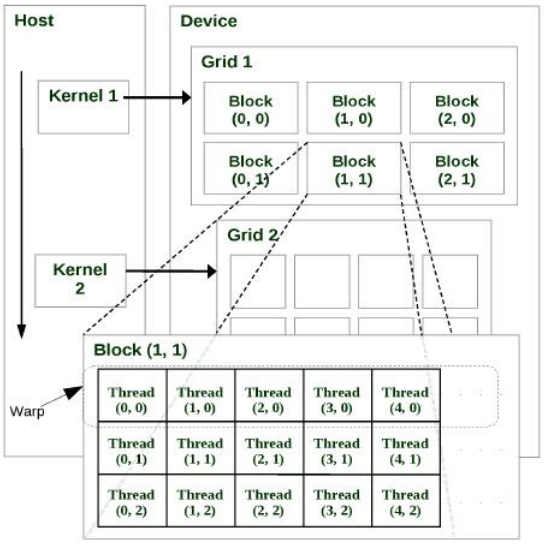


Fig. 3. NVIDIA CUDA programming model showing the sequential execution of the *host* code alongside the parallel execution of the *kernel* function on the *device* side, from [9].

Listing 1. General code flow for CUDA programming written in the CUDA extended C language

```
1 //allocate memory on GPU e.g.
2 cudaMalloc( (void**)&dev_a, N * sizeof(int)
3
4 //populate arrays
5  . . .
6
7 //copy arrays from host to device e.g.
8 cudaMemcpy( dev_a, a, N * sizeof(int),
9 cudaMemcpyHostToDevice)
10
11 //call kernel (GPU) function e.g.
12 add<<<N, 1>>>( dev_a, dev_b, dev_c );
13
14 // copy arrays from device to host e.g.
15 cudaMemcpy( c, dev_c, N * sizeof( int),
16 cudaMemcpyDeviceToHost )
17
18 //display results
19
20 //free memory e.g.
21 cudaFree( dev_a );
```

Lines 2 and 21, implement CUDA versions of the standard C language functions e.g. the standard C function *malloc* has the CUDA C function counterpart being $cudaMalloc$, and the standard C function $free$ has $cudaFree$ as its CUDA C counterpart.

Lines 8 and 15 show a CUDA C specific function, namely $cudaMemcpy$, which, given an input of pointers (from Listing 1 host code pointers are single letter variables such as a and c,while device code variable counterparts are prefixed by $dev\_$ such as dev_a and dev_c) and the size to copy (as computed by the $sizeof$ function), moves data from host to device (parameter $cudaMemcpyHostToDevice$) or device to host (parameter $cudaMemcpyDeviceToHost$).

A kernel function call uses the triple < and > operator, in this case the kernel function

$add <<< N, 1 >>>(dev\_a, dev\_b, dev\_c)$.

This function adds the values, per element (and each element is associated to 1 thread), of the variables dev_a and dev_b sent to the device, collected in variable dev_c before being sent back to the host/CPU. The variable N in this case allows the programmer to specify N number of threads which will execute the add kernel function in parallel, with 1 specifying only one block of thread for all N threads.

3.1 Design considerations for the hardware and software setup

Since the kernel function is executed in parallel in the device, the function needs to have its inputs initially moved from the CPU/host to the device, and then back from the device to the host after computation for the results. This movement of data back and forth should be minimized in order to obtain more efficient, in terms of time, execution. Implementing an equation such as (2), which involves multiplication and addition between vectors and a matrix, can be done in parallel with the previous considerations in mind. In this case, C_k, S_k, and M_Π are loaded, manipulated, and pre-processed within the host code, before being sent to the kernel function which will perform computations on these function arguments in parallel. To represent C_k, S_k, and M_Π, text files are created to house each input, whereby each element of the vector or matrix is entered in the file in order, from left to right, with a blank space in between as a delimiter. The matrix however is entered in row-major (a linear array of all the elements, rows first, then columns) order format i.e. for the matrix M_Π seen in (1), the row-major order version is simply

$$-1, 1, 1, -2, 1, 1, 1, -1, 1, 0, 0, -1, 0, 0, -2 \tag{3}$$

Row major ordering is a well-known ordering and representation of matrices for their linear as well as parallel manipulation in corresponding algorithms [13]. Once all computations are done for the $(k+1)th$ configuration, the result of equation (2) are then collected and moved from the device back to the host, where they can once again be operated on by the host/CPU. It is also important to note that these operations in the host/CPU provide logic and control of the data/inputs, while the device/GPU provides the arithmetic or computational 'muscle', the laborious task of working on multiple data at a given time in parallel, hence the current dichotomy of the CUDA programming model [9]. The GPU acts as a *co-processor* of the central processor. This division of labor is observed in Listing 1 .

3.2 Matrix computations and CPU-GPU interactions

Once all 3 initial and necessary inputs are loaded, as is to be expected from equation 2, the device is first instructed to perform multiplication between the spiking vector S_k and the matrix M_Π. To further simplify computations at this point, the vectors are treated and automatically formatted by the host code to appear as single row matrices, since vectors can be considered as such. Multiplication is done per element (one element is in one thread of the device/GPU), and then the products are collected and summed to produce a single element of the resulting vector/single row matrix.

Once multiplication of the S_k and M_Π is done, the result is added to the C_k, once again element per element, with each element belonging to one thread, executed at the same time as the others.

For this simulator, the host code consists largely of the programming language *Python*, a well-known high- level, object oriented programming (OOP) language.

The reason for using a high-level language such as Python is because the initial inputs, as well as succeeding ones resulting from exhaustively applying the rules and equation (2) require manipulation of the vector/matrix elements or values as *strings*. The strings are then concatenated, checked on (if they conform to the form (b-3) for example) by the host, as well as manipulated in ways which will be elaborated in the following sections along with the discussion of the algorithm for producing all possible and valid S_ks and C_ks given initial conditions. The host code/Python part thus implements the logic and control as mentioned earlier, while in it, the device/GPU code which is written in C executes the parallel parts of the simulator for CUDA to be utilized.

4 Simulator design and implementation

The current SNP simulator, which is based on the type of SNP systems without time delays, is capable of implementing rules of the form (b-3) i.e. whenever the regular expression E is equivalent to the regular expression a^k in that rule. Rules are entered in the same manner as the earlier mentioned vectors and matrix, as blank space delimited values (from one rule to the other, belonging to the same neuron) and $ delimited (from one neuron to the other). Thus for the SNP system Π shown earlier, the file r containing the blank space and *$* delimited values is as follows:

$$2\ 2\ \$\ 1\ \$\ 1\ 2 \tag{4}$$

That is, rule (1) from Figure 1 has the value 2 in the file r (though rule (1) isn't of the form (b-3) it nevertheless consumes a spike since its regular expression is of the same regular expression type as the rest of the rules of Π). Another implementation consideration was the use of *lists* in Python, since unlike dictionaries or tuples, lists in Python are *mutable*, which is a direct requirement of the vector/matrix element manipulation to be performed later on (concatenation mostly). Hence a $C_k =< 2, 1, 1 >$ is represented as $[2, 1, 1]$ in Python. That is, at the *kth* configuration of the system, the number of spikes of neuron 1 are given by accessing the index (starting at zero) of the configuration vector Python *list* variable $confVec$, in this case if

$$confVec = [2, 1, 1] \tag{5}$$

then $confVec[0] = 2$ gives the number of spikes available at that time for neuron 1, $confVec[1] = 1$ for neuron 2, and so on. The file r, which contains the ordered list of neurons and the rules that comprise each of them, is represented as a *list of sub- lists* in the Python/host code. For SNP system Π and from (4) we have the following:

$$r = [[2, 2], [1], [1, 2]] \tag{6}$$

Neuron 1's rules are given by accessing the sub-lists of r (again, starting at index zero) i.e. rule (1) is given by $r[0][0] = 2$ and rule (4) is given by $r[2][1] = 1$. Finally, we have the input file M, which holds the Python *list* version of (3).

4.1 Simulation algorithm implementation

The general algorithm is shown in Algorithm 1. Each line in Algorithm 1 mentions which part/s the simulator code runs in, either in the device **(DEVICE)** or in the host **(HOST)** part. Step IV of Algorithm 1 makes the algorithm stop with *2 stopping criteria* to do this:

One is when there are no more available spikes in the system (hence a zero value for a configuration vector), and the second one being the fact that all previously generated configuration vectors have been produced in an earlier time or computation, hence using them again in part I of Algorithm 1 would be pointless, since a redundant, infinite loop will only be formed.

Algorithm 1 Overview of the algorithm for the SNP system simulator

Require: Input files: $confVec$, M, r.

I. **(HOST)** Load input files. Note that M and r need only be loaded once since they are unchanging, C_0 is loaded once, and then C_ks are loaded afterwards.

II. **(HOST)** Determine if a rule/element in r is applicable based on its corresponding spike value in $confVec$, and then generate all valid and possible spiking vectors in a list of lists $spikVec$ given the 3 initial inputs.

III. **(DEVICE)** From part II., run the kernel function on $spikVec$, which contains all the valid and possible spiking vectors for the current $confVec$ and r. This will generate the succeeding C_ks and their corresponding S_ks.

IV. **(HOST+DEVICE)** Repeat steps I to IV (except instead of loading C_0 as $confVec$, use the generated C_ks in III) until a zero configuration vector (vector with only zeros as elements) or further C_ks produced are repetitions of a C_k produced at an earlier time. (Stopping criteria in subsection 4.1)

Another important point to notice is that either of the stopping criterion from 4.1 could allow for a deeply nested computation tree, one that can continue executing for a significantly lengthy amount of time even with a multi-core CPU and even the more parallelized GPU.

4.2 Closer inspection of the SN P system simulator

The more detailed algorithm for part II of Algorithm 1 is as follows.

Recall from the definition of an SNP system (Definitin 1) that we have m number of σs. We related m to our implementation by noticing the cardinality of the Python list r.

$$|r| = m \tag{7}$$

$$\Psi = |\sigma_{V_1}||\sigma_{V_2}|...|\sigma_{V_m}| \tag{8}$$

where

$|\sigma_{V_m}|$

means the total number of rules in the mth neuron which satisfy the regular expresion E in (b-3). m gives the total number of neurons, while Ψ gives the expected number of *valid* and *possible* S_ks which should be produced in a given configuration. We also define ω as both the largest and last integer value in the sub-list (neuron) created in step II of Algorithm 1 and further detailed in Algorithm 2, which tells us how many elements of that neuron satisfy E.

During the exposition of the algorithm, the previous Python lists (from their vector/matrix counterparts in earlier sections) (5) and (6) will be utilized. For part II Algorithm 1 we have a sub-algorithm (Algorithm 2) for generating all valid and possible spiking vectors given input files M, $confVec$, and r.

Algorithm 2 Algorithm further detailing part II in Algorithm 1

II-1. Create a list *tmp*, a copy of r, marking each element of *tmp* in increasing order of $\mathbb{N}$, as long as the element/s satisfy the rule's regular expression E of a rule (given by list r). Elements that don't satisfy E are marked with 0.

II-2. To generate all possible and valid spiking vectors from *tmp*, we go through each neuron i.e. all elements of *tmp*, since we know a priori m as well as the number of elements per neuron which satisfy E. We only need to iterate through each neuron/element of *tmp*, ω times. (from II-1). We then produce a new list, *tmp*2, which is made up of a sub-list of strings from all possible and valid {*1,0*} strings i.e. spiking vectors per neuron.

II-3. To obtain all possible and valid {*1,0*} strings (S_ks), given that there are multiple strings to be concatenated (as in *tmp*2's case), pairing up the neurons first, in order, and then exhaustively distributing every element of the first neuron to the elements of the 2nd one in the pair. These paired-distributed strings will be stored in a new list, *tmp*3.

Algorithm 2 ends once all {*1,0*} have been paired up to one another. As an illustration of Algorithm 2, consider (5), (6), and (1) as inputs to our SNP system simulator. The following details the production of all valid and possible spiking vectors using Algorithm 2.

Initially from II-1 of Algorithm 2, we have

$r = tmp = [[2, 2], [1], [1, 2]]$.

Proceeding to illustrate II-2 we have the following passes.

1st pass: $tmp = [[1, 2], [1], [1, 2]]$

Remark/s: previously, $tmp[0][0]$ was equal to 2, but now has been changed to 1, since it satisfies E ($configVec[0] = 2$ w/c is equal to 2, the number of spikes consumed by that rule).Σ

2nd pass: $tmp = [[1, 2], [1], [1, 2]]$

Remark/s: previously $tmp[0][1] = 2$, which has now been changed (incidentally) to 2 as well, since it's the 2nd element of σ_1 which satisfies E.

3rd pass: $tmp = [[1, 2], [1], [1, 2]]$

Remark/s: 1st (and only) element of neuron 2 which satisfies E.

4th pass: $tmp = [[1, 2], [1], [1, 2]]$

Remark/s: Same as the 1st pass

5th pass: $tmp = [[1, 2], [1], [1, 0]]$

Remark/s: element $tmp[2][1]$, or the 2nd element/rule of neuron 3 doesn't satisfy E.

Final result: $tmp = [[1, 2], [1], [1, 0]]$

At this point we have the following, based on the earlier definitions:

$m = 3$ (3 neurons in total, one per element/value of $confVec$)

$\Psi = |\sigma_{V_1}||\sigma_{V_2}||\sigma_{V_3}| = 2 * 1 * 1 = 2$

Ψ tells us the number of valid strings of *1*s and *0*s i.e. S_ks, which need to be produced later, for a given C_k which in this case is $confvec$. There are only 2 valid S_ks/spiking vectors from (5) and the rules given in (6) encoded in the Python list r. These S_ks are

$$< 0, 1, 1, 1, 0 > \tag{9}$$

$$< 1, 0, 1, 1, 0 > \tag{10}$$

In order to produce all S_ks in an algorithmic way as is done in Algorithm 2 , it's important to notice that first, *all possible and valid* S_ks (made up of *1*s and *0*s) per σ have to be produced first, which is facilitated by II-1 of Algorithm 2 and its output (the current value of the list tmp).

Continuing the illustration of II-1, and illustrating II-2 this time, we iterate over neuron 1 twice, since its $\omega = 2$, i.e. neuron 1 has only 2 elements which satisfy E, and consequently, it is its 2nd element,

$tmp[0][1] = 2$.

For neuron 1, our first pass along its elements/list is as follows. Its 1st element,

$tmp[0][0] = 1$

is the first element to satisfy E, hence it requires a *1* in its place, and *0* in the others. We therefore produce the string '*10*' for it. Next, the 2nd element satisfies E and it too, deserves a *1*, while the rest get *0*s. We produce the string '*01*' for it.

The new list, $tmp2$, collecting the strings produced for neuron 1 therefore becomes

$tmp2 = [[10, 01]]$

Following these procedures, for neuron 2 we get $tmp2$ to be as follows:

$tmp2 = [[10, 01], [1]]$

Since neuron 2 which has only one element only has 1 possible and valid string, the string 1. Finally, for neuron 3, we get $tmp2$ to be

$tmp2 = [[10, 01], [1], [10]]$

In neuron 3, we iterated over it only once because ω, the number of elements it has which satisfy E, is equal to 1 only. Observe that the sublist

$tmp2[0] = [10, 01]$

is equal to all possible and valid {*1,0*} strings for neuron 1, given rules in (6) and the number of spikes in $configVec$.

Illustrating II-3 of Algorithm 2, given the valid and possible {*1,0*} strings (spiking vectors) for neurons 1, 2, and 3 (separated per neuron-column) from (5)

and (6) and from the illustration of II-2, all possible and valid list of $\{1,0\}$ string/s for neuron 1: ['10','01'], neuron 2: ['1'], and neuron 3: ['10'].

First, pair the strings of neurons 1 and 2, and then distribute them exhaustively to the other neuron's possible and valid strings, concatenating them in the process (since they are considered as *strings* in Python).

'10' + '1' → '101'
'01'
and
'10'
'01' + '1' → '011'

now we have to create a new list from $tmp2$, which will house the concatenations we'll be doing. In this case,

$tmp3 = [101, 011]$

next, we pair up $tmp3$ and the possible and valid strings of neuron 3

'101' + '10' → '10110'
'011'
and
'101'
'011' + '10' → '01110'

eventually turning $tmp3$ into

$tmp3 = [10110, 01110]$

The final output of the sub-algorithm for the generation of all valid and possible spiking vectors is a list,

$tmp3 = [10110, 01110]$

As mentioned earlier, $\Psi = 2$ is the number of valid and possible S_ks to be expected from r, M_Π, and C_0 = [2,1,1] in Π. Thus $tmp3$ is the list of all possible and valid spiking vectors given (5) and (6) in this illustration. Furthermore, $tmp3$ includes all possible and valid spiking vectors for a given neuron in a given configuration of an SN P system with all its rules and synapses (interconnections). Part II-3 is done ($m - 1$) times, albeit exhaustively still so, between the two lists/neurons in the pair.

5 Simulation results, observations, and analyses

The SNP system simulator (combination of Python and CUDA C) implements the algorithms in section 4 earlier. A sample simulation run with the SNP system Π is shown below (most of the output has been truncated due to space constraints) with C_0 = [2,1,1]

```
****SN P system simulation run STARTS here****
Spiking transition Matrix:
...

Rules of the form a^n/a^m -> a or a^n ->a loaded:
```

```
['2', '2', '$', '1', '$', '1', '2']

Initial configuration vector: 211

Number of neurons for the SN P system is 3
Neuron 1  rules criterion/criteria and total order
...

 tmpList =  [['10', '01'], ['1'], ['10']]
 All valid spiking vectors: allValidSpikVec =
 [['10110', '01110']]
All generated Cks are allGenCk =
['2-1-1', '2-1-2', '1-1-2']
 End of C0
**
**
**
 initial total Ck list is
   ['2-1-1', '2-1-2', '1-1-2']
Current confVec: 212
All generated Cks are allGenCk =
['2-1-1', '2-1-2', '1-1-2', '2-1-3', '1-1-3']
**
**
**
Current confVec: 112
All generated Cks are allGenCk =
['2-1-1', '2-1-2', '1-1-2', '2-1-3', '1-1-3',
'2-0-2', '2-0-1']
**
**
...

Current confVec: 109
All generated Cks are allGenCk = ['2-1-1', '2-1-2',
...
 '1-0-7', '0-1-9', '1-0-8', '1-0-9']

**
**
**

No more Cks to use (infinite loop/s otherwise). Stop.
****SN P system simulation run ENDS here****
```

That is, the computation tree for SNP system Π with C_0 = [2,1,1] went down as deep as $confVec = 109$. At that point, all configuration vectors for all possible and valid spiking vectors have been produced. The Python list variable $allGenCk$ collects all the C_ks produced. In Algorithm 2 all the values of $tmp3$ are added to $allGenCk$. The final value of $allGenCk$ for the above simulation run is

allGenCk = ['2-1-1', '2-1-2', '1-1-2', '2-1-3', '1-1-3', '2-0-2', '2-0-1', '2-1-4', '1-1-4', '2-0-3', '1-1-1', '0-1-2', '0-1-1', '2-1-5', '1-1-5', '2-0-4', '0-1-3', '1-0-2', '1-0-1', '2-1-6', '1-1-6', '2-0-5', '0-1-4', '1-0-3', '1-0-0', '2-1-7', '1-1-7', '2-0-6', '0-1-5', '1-0-4', '2-1-8', '1-1-8', '2-0-7', '0-1-6', '1-0-5', '2-1-9', '1-1-9', '2-0-8', '0-1-7', '1-0-6', '2-1-10', '1-1-10', '2-0-9', '0-1-8', '1-0-7', '0-1-9', '1-0-8', '1-0-9']

It's also noteworthy that the simulation for Π didn't stop at the 1st stopping criteria (arriving at a zero vector i.e. C_k = [0,0,0]) since Π generates all natural counting numbers greater than 1, hence a loop (an infinite one) is to be expected. The simulation run shown above stopped with the 2nd stopping criteria from Section 4. Thus the simulation was able to exhaust all possible configuration vectors and their spiking vectors, stopping only since a repetition of an earlier generated $confVec/C_k$ would introduce a loop (triggering the 2nd stopping criteria in subsection 4.1). Graphically (though not shown exhaustively) the computation tree for Π is shown in Figure 4.

The $confVecs$ followed by (...) are the $confVecs$ that went deeper i.e. produced more C_ks than Figure 4 has shown.

6 Conclusions and future work

Using a highly parallel computing device such as a GPU, and the NVIDIA CUDA programming model, an SNP system simulator was successfully designed and implemented as per the objective of this work. The simulator was shown to model the workings of an SN P system without delay using the system's matrix representation. The use of a high level programming language such as Python for host tasks, mainly for logic and string representation and manipulation of values (vector/matrix elements) has provided the necessary expressivity to implement the algorithms created to produce and exhaust all possible and valid configuration and spiking vectors. For the device tasks, CUDA allowed the manipulation of the NVIDIA CUDA enabled GPU which took care of repetitive and highly parallel computations (vector-matrix addition and multiplication essentially).

Future versions of the SNP system simulator will focus on several improvements. These improvements include the use of an optimized algorithm for matrix computations on the GPU without requiring the input matrix to be transformed into a square matrix (this is currently handled by the simulator by padding zeros to an otherwise non-square matrix input). Another improvement would be the simulation of systems not of the form (b-3). Byte-compiling the Python/host

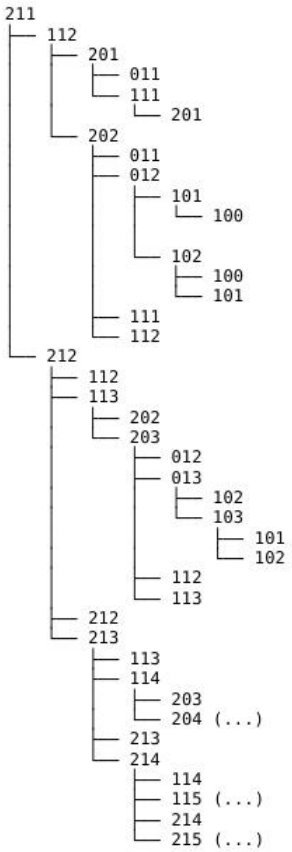


Fig. 4. The computation tree graphically representing the output of the simulator run over Π with $C_0 = [2, 1, 1]$

part of the code to improve performance as well as metrics to further enhance and measure execution time are desirable as well. Finally, deeper understanding of the CUDA architecture, such as inter- thread/block communication, for very large systems with equally large matrices, is required. These improvements as well as the current version of the simulator should also be run in a machine or setup with higher versions of GPUs supporting NVIDIA CUDA.

7 Acknowledgments

Francis Cabarle is supported by the DOST-ERDT scholarship program. Henry 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. They would also like to acknowledge the Algorithms and

Complexity laboratory for the use of Apple iMacs with NVIDIA CUDA enabled GPUs for this work. Miguel 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 the support of the project TIN2009–13192 of the "Ministerio de Educación y Ciencia" of Spain, both co-financed by FEDER funds. Finally, they would also like to thank the valuable insights of Mr. Neil Ibo.

References

1. M. Harris, "Mapping computational concepts to GPUs", *ACM SIGGRAPH 2005 Courses*, NY, USA, 2005.
2. M. Gross, "Molecular computation", *Chapter 2 of Non-Standard Computation*, (T. Gramss, S. Bornholdt, M. Gross, M. Mitchel, Th. Pellizzari, eds.), Wiley-VCH, Weinheim, 1998.
3. M. Ionescu, Gh. Păun, T. Yokomori, "Spiking Neural P Systems", *Journal Fundamenta Informaticae* , vol. 71, issue 2,3 pp. 279-308, Feb. 2006.
4. X. Zeng, H. Adorna, M. A. Martínez-del-Amor, L. Pan, "When Matrices Meet Brains", *Proceedings of the Eighth Brainstorming Week on Membrane Computing* , Sevilla, Spain, Feb. 2010.
5. X. Zeng, H. Adorna, M. A. Martínez-del-Amor, L. Pan, M. Pérez-Jiménez, "Matrix Representation of Spiking Neural P Systems", *11th International Conference on Membrane Computing* , Jena, Germany, Aug. 2010.
6. Gh. Păun, G. Ciobanu, M. Pérez-Jiménez (Eds), *"Applications of Membrane Computing"* , Natural Computing Series, Springer, 2006.
7. P systems resource website. (2011, Jan) [Online]. Available: `www.ppage.psystems.eu`.
8. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, "Simulating a P system based efficient solution to SAT by using GPUs", *Journal of Logic and Algebraic Programming*, Vol 79, issue 6, pp. 317-325, Apr. 2010.
9. 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*, Vol 11, issue 3, pp. 313-322, Mar. 2010.
10. D. Díaz, C. Graciani, M.A. Gutiérrez, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Software for P systems. In Gh. Păun, G. Rozenberg, A. Salomaa (eds.) *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford (U.K.), Chapter 17, pp. 437-454, 2009.
11. G. Ciobanu, G. Wenyuan. P Systems Running on a Cluster of Computers. *Lecture Notes in Computer Science*, 2933, 123-139, 2004.
12. V. Nguyen, D. Kearney, G. Gioiosa. A Region-Oriented Hardware Implementation for Membrane Computing Applications and Its Integration into Reconfig-P. *Lecture Notes in Computer Science*, 5957, 385-409, 2010.
13. D. Kirk, W. Hwu, *"Programming Massively Parallel Processors: A Hands On Approach"* , 1st ed. MA, USA: Morgan Kaufmann, 2010.
14. NVIDIA corporation, *"NVIDIA CUDA C programming guide"* , version 3.0, CA, USA: NVIDIA, 2010.
15. NVIDIA CUDA developers resources page: tools, presentations, whitepapers. (2010, Jan) [Online]. Available: `http://developer.nvidia.com/page/home.html`

16. V. Volkov, J. Demmel, "Benchmarking GPUs to tune dense linear algebra", *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, NJ, USA, 2008.
17. K. Fatahalian, J. Sugerman, P. Hanrahan, "Understanding the efficiency of GPU algorithms for matrix-matrix multiplication", *In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (HWWS '04)* , ACM, NY, USA, pp. 133-137, 2004

Designing Tissue-like P Systems for Image Segmentation on Parallel Architectures

Javier Carnero[1], Daniel Díaz-Pernil[1], Miguel A. Gutiérrez-Naranjo[2]

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

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

Summary. Problems associated with the treatment of digital images have several interesting features from a bio-inspired point of view. One of them is that they can be suitable for parallel processing, since the same sequential algorithm is usually applied in different regions of the image. In this paper we report a work-in-progress of a hardware implementation in *Field Programmable Gate Arrays* (FPGAs) of a family of tissue-like P systems which solves the segmentation problem in digital images.

1 Introduction

Membrane Computing is a computational paradigm inspired in the functioning of living cells and tissues. One of its characteristic features is the use of parallelism as a computation tool. In many of the models, the devices perform the computation by applying parallelization in a double sense: on the one hand, several rules can be applied simultaneously in each membrane; on the other hand, all the membranes perform the computation at the same time.

In spite of recent efforts [15], it seems that in the next future there will not be an implementation of P systems *in vivo* or *in vitro*. All the possible approaches to the theoretical model lean on the current computer architectures.

In this line, many efforts have been made for obtaining a *simulation* of the P system behavior with current computers [13, 16]. Most of these simulators are thought for running on one-processor computers. These sequential machines only perform one action per time unit and the parallelism of the membrane computing devices is lost. This bottle-neck produces a serious discrepancy between the theo-

retical efficiency of the P systems and the realistic resources needed for performing a computation.

In the last years, according with the development of new parallel architectures, new attempts have been made for approaching the computation of P systems by performing several actions in the same step. This does not mean a real implementation of the P system, but it can be considered as a new step toward a more realistic simulation.

The first parallel and distributed simulators were presented in 2003. In [12], a parallel implementation of transition P systems was presented. The program was designed for a cluster of 64 dual processor nodes and it was implemented and tested on a Linux cluster at the National University of Singapore. In [32], a purely distributive simulator of P systems was presented. It was implemented using Java's *Remote Methods Invocation* to connect a number of computers that interchange data. The class of P systems that the simulator can accept is a subset of the $NOP_2(coo, tar)$ family of systems, which have the computational power of Turing machines.

Also in 2003, Petreska and Teuscher [29] presented a parallel hardware implementation of a special class of membrane systems. The implementation was based on a universal membrane hardware component that allows efficiently run P system on a reconfigurable hardware known as *Field Programmable Gate Arrays* (FPGAs) [35]. Recently, a new research line has arisen due to a novel device architecture called CUDA^{TM}, (Compute Unified Device Architecture) [39]. It is a general purpose parallel computing architecture that allows the parallel compute engine in NVIDIA Graphic Processor Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU [5, 6, 7]. Following the research line started in [29], Van Nguyen *et al.* have proposed the use hardware implementation for membrane computing applications [22, 23, 24, 25] based on reconfigurable computing technology called Reconfig-P.

In this paper, we also explore the possibilities of the *Field Programmable Gate Arrays* (FPGAs) for building a hardware implementation of P systems. The P system model chosen for the implementation has been tissue-like P systems and as a case study we consider the *segmentation problem* in 2D images.

Segmentation in computer vision (see [31]), refers to the process of partitioning a digital image into multiple segments (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. Technically, the process consists on assigning a label to each pixel, in such way that pixels with the same label form a meaningful region. There exist different techniques to segment an image. Some techniques are *clustering methods* [1, 36], *histogram-based methods* [34], *Watershed transformation methods* [33], image pyramids methods

[18] or *graph partitioning methods* [37, 38]. Some of the practical applications of image segmentation are medical imaging [36] or face recognition [17].

Segmentation in Digital Imagery has several features which make it suitable for techniques inspired by nature. One of them is that it can be paralleled and locally solved. Regardless how large is the picture, the segmentation process can be performed in parallel in different local areas of it. Another interesting feature is that the basic necessary information can be easily encoded by bio-inspired representations.

In the literature, one can find several attempts for bridging problems from Digital Imagery with Natural Computing as the works by K.G. Subramanian *et al.* [8, 9] or the work by Chao and Nakayama where Natural Computing and Algebraic Topology are linked by using Neural Networks [10] (extended Kohonen mapping). In this paper, we will use an information encoding and techniques borrowed from Membrane Computing. This paper is a new step in the research started at [4], where the authors present an implementation of a membrane solution of a segmentation problem using hardware programming. In this paper, we present a different family of tissue-like P systems to solve the problem and report the hardware implementation. In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems[3].

The paper is organized as follows: firstly, we present our bio-inspired formal framework. Next, we present a family of tissue-like P systems designed to obtain an edge-based segmentation of a 2D digital image. Then, general considerations about designing hardware P systems are studied, focusing on the segmentation problem. The paper finishes with some conclusions and future work.

2 Formal Framework: Tissue-like P Systems

Tissue-like P systems were presented by Martín–Vide *et al.* in [21]. They have two biological inspirations (see [20]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a network of processors dealing with symbols and communicating these symbols along channels specified in advance.

The main features of this model, from the computational point of view, are that cells do not have polarization and the membrane structure is a general graph.

Formally, a *tissue-like P system* with input of degree $q \geq 1$ is a tuple

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

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*;
2. $\Sigma (\subset \Gamma)$ is the input alphabet;

[3] We refer to [26] for basic information in this area, to [28] for a comprehensive presentation and the web site [40] for the up-to-date information.

3. $\mathcal{E} \subseteq \Gamma$ (the objects in the environment);
4. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration;
5. $\mathcal{R}$ is a finite set of communication rules of the following form:

$$(i, u/v, j)$$

for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;
6. $i_\Pi \in \{1, 2, \ldots, q\}$ is the input cell;
7. $o_\Pi \in \{0, 1, 2, \ldots, q\}$ is the output cells

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

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

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

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

A *configuration* is an instantaneous description of the P system Π. Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A sequence of computation steps is called a *computation*. A configuration is *halting* when no rules can be applied to it. Then, a computation halts when the P system reaches a halting configuration.

3 Segmenting Digital Images

A *point set* is simply a topological space consisting of a collection of objects called points and a topology which provides for such notions as *nearness* of two points, the *connectivity* of a subset of the point set, the *neighborhood* of a point, *boundary points*, and *curves* and *arcs*.

The most common point sets occurring in image processing are discrete subsets of N-dimensional Euclidean space $\mathbb{R}^n$ with $n = 1, 2$ or 3 together with the discrete topology. There is no restriction on the shape of the discrete subsets of $\mathbb{R}^n$ used in applications of image algebra to solve vision problems.

For a point set X in Z, a *neighborhood function* from X in Z, is a function $N : X \to 2^Z$. For each point $x \in X$, $N(x) \subseteq Z$. The set $N(x)$ is called a *neighborhood* for x.

There are two neighborhood function on subsets of $\mathbb{Z}^2$ which are of particular importance in image processing, the *von Neumann* neighborhood and the *Moore* neighborhood. The first one $N : X \to 2^{\mathbb{Z}^2}$ is defined by $N(x) = \{y : y = (x_1 \pm j, x_2) \text{ or } y = (x_1, x_2 \pm k),\ j,k \in \{0,1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbb{Z}^2$. While the Moore neighborhood $M : X \to 2^{\mathbb{Z}^2}$ is defined by $M(x) = \{y : y = (x_1 \pm j, x_2 \pm k),\ j,k \in \{0,1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbb{Z}^2$. The von Neumann and Moore neighborhood are also called the *four neighborhood* (4-adjacency) and *eight neighborhood* (8-adjacency), respectively. In this paper, we work with 4-adjacency. The point sets with the usual operations has an algebra structure (see [30]).

An *Z-valued image* on X is any element of Z^X. Given an Z-valued image $I \in Z^X$, i.e. $I : X \to Z$, then Z is called the set of possible range values of I and X the spatial domain of I. The graph of an image is also referred to as the *data structure representation* of the image. Given the data structure representation $I = \{(x, I(x)) : x \in X\}$, then an element $(x, I(x))$ is called a *picture element* or *pixel*. The first coordinate x of a pixel is called the *pixel location* or *image point*, and the second coordinate $I(x)$ is called the *pixel value* of I at location x.

For example, X could be a subset of $\mathbb{Z}^2$ where $x = (i, j)$ denotes spatial location, and Z could be a subset of $\mathbb{N}$, $\mathbb{N}^3$, etc. So, given an image $I \in Z^{\mathbb{Z}^2}$, a pixel of I is the form $((i, j), I(x))$, which will be denoted by $I(x)_{ij}$. We call the *set of colors* or *alphabet of colors* to the image set of the function I with domain X and the image point of each pixel is called *associated color*. We can consider an order in this set. In this paper, we denote Z as $\mathcal{C}_I$. Usually, we consider in digital image a predefined alphabet of colors $\mathcal{C}$. We define $h = |\mathcal{C}|$ as the size (number of colors) of $\mathcal{C}$. In this paper, we work with images in grey scale, then $\mathcal{C} = \{0, \ldots, 255\}$, where 0 codify the black color and 255 the white color.

By technical reasons, we use below different ways to codify a same pixel. For example, if we take the pixel $((i, j), a)$ we could codify with the following expressions: a_{ij}, A_{ij}, a'_{ij}, $\overline{a}_{ij}$, $(a, l)_{ij}$ with $l \in \mathbb{N}$, etc.

A *region* could be defined by a subset of the domain of I whose points are all mapped to the same (or similar) pixel value by I. So, we can consider the region R_i as the set $\{x \in X : I(x) = i\}$ but this kind of regions has not to be connected. We prefer to consider a region r as a maximal connected subset of a set like R_i. We say two regions r_1, r_2 are adjacent when at less a pair of pixel $x_1 \in r_1$ and $x_2 \in r_2$ are adjacent. We say x_1 and x_2 are *border pixels*. If $I(x_1) < I(x_2)$ we say x_1 is an *edge pixel*. The set of connected edge pixels with the same pixel value is called a *boundary* between two regions.

From a general point of view, segmentation refers to the process of partitioning a digital image into multiple regions. Thresholding is a method of image segmentation whose basic aim is to obtain a binary image from a colour one. The idea is to split the set of pixels into two sets (black and white) depending on its bright and a fixed valued, the *threshold*. If the bright of the pixel is greater than the threshold,

then the pixel is labelled as *object*. Otherwise, it is labelled as *background*. After labelling, a new binary image is created by colouring each pixel white or black, depending on the label.

The basic thresholding method can be generalized in a natural way. Instead of getting a binary image by labelling the original set of pixels by $\{0, 1\}$, we can consider a larger set of labels, $\{1, \ldots, k\}$ so we obtain a final image with k levels. Another natural generalization is to replace the colour information by another scale on the features of the pixel (bright, intensity, gray scale, etc.).

Edge detection is an important operation in a large number of image processing applications, such as image segmentation, character recognition and scene analysis.

In this paper we work with the first one, the *edge-based segmentation of 2D digital images problem (2D-ES problem)*, which is described as follows: *Given a digital 2D image with pixels of (possibly) different colors, obtain the boundaries of regions in that image.*

In order to provide a logarithmic-time uniform solution to our problem, we design a family of tissue-like P systems, Π. Given an image I of size n^2, we take the P system $\Pi(n, k)$ of the family to work with I. The input data (image I) is codified by a set of objects a'_{ij}, with $a \in \mathcal{C}$ and $1 \leq i, j \leq n$ and k is referred to the number of processing cells. So, when we work with a parallel architecture we do not have to know previously an exact number of processors to work. Then, we introduce the parameter k to solve this problem.

The functioning of a P system of the family consists of the following stages:

- First of all, the P system generates 8 auxiliary copies of the input data. Then, we have 9 codifications of the input image, but one of them is distinguished of the rest. So, we can work with each pixel without taking into account what happens with the rest of the image.
- Second, the P system applies a *basic noise filter* in order to eliminate some pickle noise that could affect the segmentation process. The P system will apply the largely used average filter because of its simplicity and good results. For each pixel, the process consists of calculating the average average of its adjacent pixels. If the distance between the pixel and its average is greater than a threshold ρ, the pixel will be considered as noise and it will be replaced by its average colour.
- Next, the P system performs a thresholding of the image to solve the problem of degradation of colours of pixels in the boundary of adjacent regions with different colours.
- Once this process is finished, the P system applies a translation of rules defined in [11] obtaining an edge-based segmentation of the image took of the previous stage.

The family $\mathbf{\Pi} = \{\Pi(n, k) : n, k \in \mathbb{N}\}$ of tissue-like P systems of degree $k + 1$ is defined as follows:

For each $n, k \in \mathbb{N}$,

$$\Pi(n, k) = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_{k+1}, \mathcal{R}, i_\Pi, o_\Pi),$$

defined as follows:

- $\Gamma = \Sigma \cup \{a_{ij}, a''_{ij}, \bar{a}_{ij}, A_{ij}, A'_{ij}, A''_{ij}, \overline{A}_{ij}, \overline{\overline{A}}_{ij}, (a,1)_{ij}, (a,2)_{ij}, (a,3)_{ij} \ : \ 1 \leq i,j \leq n,\ a \in \mathcal{C}\}$ is the working alphabet;
- the input alphabet is $\Sigma = \{a'_{ij} : 1 \leq i,j \leq n,\ a \in \mathcal{C}, I(i,j) = a\}$;
- the environment alphabet is $\mathcal{E} = \Gamma \setminus \Sigma$;
- the multisets of the cells are $w_1 = \{\{\nu^3_{ij}, \nu^3_{ji} \ : \ i = 0, n+1, 0 \leq j \leq n+1\}\}$, $w_2 = \cdots = w_{k+1} = T^{\lceil n^2/k \rceil}$, respectively. We call to the last k cells as *processing cells*;
- R is the following set of communication rules:

1. $(1, a'_{ij}/a^8_{ij} A_{ij}, 0)$
   for $1 \leq i,j \leq n$.
   These rules are used to generate new elements, so the P system can work in parallel with each pixel and forget what happen with the rest of the image. The P system first uses these elements to work with the noise of our image.

2. $$\left(\begin{array}{lll} & c_{i-1j-1} & d_{i-1j} \ e_{i-1j+1} \\ 1, & b_{ij-1} \quad A_{ij} & f_{ij+1} \end{array} \ / \ T, t \right)$$
   $$\begin{array}{lll} & & \\ & l_{i+1j-1} & h_{i+1j} \ g_{i+1j+1} \end{array}$$
   for
   - $1 \leq i,j \leq n$,
   - $a,b,c,d,e,f,g,h,l \in \mathcal{C} \cup \{\nu\}$.

   This type of rules are used to translate each object A_{ij} and one copy of their neighbours (objects) to a processing cell. We are sure that all the pixels not go to the same cell, because our P system has n^2 or n^2+1 objects T spread over processing cells, each one with a similar number of copies of T.

3. $$\left(\begin{array}{lll} & c_{i-1j-1} & d_{i-1j} \ e_{i-1j+1} \\ t, & b_{ij-1} \quad A_{ij} & f_{ij+1} \end{array} \ / \ \alpha'_{ij}, 0 \right)$$
   $$\begin{array}{lll} & & \\ & l_{i+1j-1} & h_{i+1j} \ g_{i+1j+1} \end{array}$$
   for
   - $1 \leq i,j \leq n$,
   - $a,b,c,d,e,f,g,h,l \in \mathcal{C} \cup \{\nu\}$,
   - We take μ as the number of pixels with colors in $\mathcal{C}$ and $\nu = 0$. Then, $av(a) = (b+c+d+e+f+g+h+i)/\mu$,
   - α is the nearest colour in $\mathcal{C}$ to the average colour $av(a)$ with $|\alpha - av(a)| > \rho$, with $\rho \in \mathbb{R}$.

4. $$\left(\begin{array}{lll} & c_{i-1j-1} & d_{i-1j} \ e_{i-1j+1} \\ t, & b_{ij-1} \quad A_{ij} & f_{ij+1} \end{array} \ / \ a'_{ij}, 0 \right)$$
   $$\begin{array}{lll} & & \\ & i_{i+1j-1} & h_{i+1j} \ g_{i+1j+1} \end{array}$$
   for
   - $1 \leq i,j \leq n$,
   - $a,b,c,d,e,f,g,h,i \in \mathcal{C} \cup \{\nu\}$,

- We take μ as the number of pixels with colors in $\mathcal{C}$ and $\nu = 0$. Then, $av(a) = (b + c + d + e + f + g + h + i)/\mu$,
- $|a - av(a)| \leq \rho$, where $\rho_1 \in \mathbb{R}$.

This set of rules is used to detect the noise and correct it with the average colour of its adjacent pixels. We find here a local thresholding (with respect to the colors) with predefined threshold ρ. In fact, we are simulating one of the more typical algorithms to remove noise. The P system changes the notation of the objects which are codifying pixels and they adopt the form a'_{ij}, with $a \in \mathcal{C}$.

5. $(t, b'_{ij}/A'_{ij}, 0)$
   for
   - $1 \leq i, j \leq n$,
   - $\tau = (|\mathcal{C}|/\rho_2)$, $l = 0, 1, 2, \ldots, \rho_2$,
   - If $b \in \mathcal{C}$ then $a \in \mathcal{C}$ $(a < b \leq a + (\tau - 1)$ and $a = \tau \cdot l)$ or $(b = a = \tau \cdot l)$,
   - If $b = \nu$ then $A = \nu$.

   These rules are used to discretize the colors dividing the set of colors in ρ_2 subsets of length ν. We find here a general thresholding (with respect to the colors) with predefined threshold ν.

6. $(t, A'_{ij}/T, 1)$
   for
   - $0 \leq i, j \leq n + 1$, $2 \leq t \leq k + 1$,
   - $a \in \mathcal{C}$.

   This set of rules are used to send our transformed image to the cell 1. Now, the objects A'_{ij} encode the pixels of our image.

7. $(1, A'_{ij}/A''_{ij}\overline{A}_{ij}\overline{a}^8_{ij}, 0)$
   for
   - $0 \leq i, j \leq n + 1$,
   - $a \in \mathcal{C} \cup \{\nu\}$.

   The P system uses these rules to generate enough copies of our image to perform the segmentation process in the cells $2, \ldots, k$ and $k+1$. The objects A''_{ij} are used in the second part of the segmentation. The rest of the objects are used in the first part of the segmentation.

8. $$\left(t, \begin{array}{ccc} \overline{c}_{i-1j-1} & \overline{d}_{i-1j} & \overline{e}_{i-1j+1} \\ \overline{b}_{ij-1} & \overline{A}_{ij} & \overline{f}_{ij+1} \\ \overline{i}_{i+1j-1} & \overline{h}_{i+1j} & \overline{g}_{i+1j+1} \end{array} / \ T, t \right)$$
   for
   - $1 \leq i, j \leq n$,
   - and $a, b, c, d, e, f, g, h, i \in \mathcal{C} \cup \{\nu\}$.

   These rules are defined to send new objects to the processing cells to do the first part of the segmentation. We look for edge pixels.

9. $(t, \overline{A}_{ij}\overline{b}_{kl} / \overline{\overline{A}}_{ij}\overline{b}_{kl}, 0)$,
for
   - $1 \leq i, j, k, l \leq n$, $(i, j), (k, l)$ adjacent pixels,
   - $a, b \in \mathcal{C}$ and $a < b$.

   These rules are used to mark edge pixels. In fact, the the P system brings from the environment an object of the form $\overline{\overline{A}}_{ij}$ for each edge pixel. Our problem is the edge pixels not always are adjacent. So, we do not have an only one set of connected edge pixel forming a boundary. Then, we should add the necessary pixel to connect all the edge pixels of a boundary.
10. $(t, \overline{\overline{A}}_{ij} / T, 1)$
for
   - $1 \leq i, j \leq n$,
   - $a \in \mathcal{C}$.

   These rules send the edge pixels to the cell 1.
11. $(1, \overline{\overline{A}}_{ij} / (a, 1)^2_{ij}, 0)$
for
   - $0 \leq i, j \leq n + 1$,
   - $a \in \mathcal{C} \cup \{\nu\}$.

   The P system uses these rules to generate two copies of our edge pixels to perform the second part of the segmentation in processing cells.
12. $(1, A''_{ij} / (a, 2)^2_{ij}, 0)$
for
   - $0 \leq i, j \leq n + 1$,
   - $a \in \mathcal{C} \cup \{\nu\}$.

   The P system uses these rules to generate enough copies of our image to perform the second part of the segmentation in processing cells.
13. $\left(1, \begin{matrix}(a,1)_{i-1j-1} & (a,2)_{i-1j} \\ (b,2)_{ij-1} & (a,1)_{ij}\end{matrix} \;/\; T, t\right) \left(1, \begin{matrix}(b,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (a,2)_{ij}\end{matrix} \;/\; T, t\right)$
$\left(1, \begin{matrix}(a,1)_{i-1j-1} & (b,2)_{i-1j} \\ (a,2)_{ij-1} & (a,1)_{ij}\end{matrix} \;/\; T, t\right) \left(1, \begin{matrix}(a,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (b,2)_{ij}\end{matrix} \;/\; T, t\right)$
for
   - $1 \leq i, j \leq n$,
   - $a, b \in \mathcal{C}$.

   These rules are defined to send new objects to the processing cells to do the second part of the segmentation. We look for new edge pixels.
14. $\left(1, \begin{matrix}(a,1)_{i-1j-1} & (a,2)_{i-1j} \\ (b,2)_{ij-1} & (a,1)_{ij}\end{matrix} \;/\; \begin{matrix}(a,3)_{i-1j-1} & (a,3)_{i-1j} \\ (b,2)_{ij-1} & (a,3)_{ij}\end{matrix}, t\right)$
$\left(1, \begin{matrix}(b,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (a,2)_{ij}\end{matrix} \;/\; \begin{matrix}(b,2)_{i-1j-1} & (a,3)_{i-1j} \\ (a,3)_{ij-1} & (a,3)_{ij}\end{matrix}, t\right)$

$$\left(1, \begin{matrix}(a,1)_{i-1j-1} & (b,2)_{i-1j} \\ (a,2)_{ij-1} & (a,1)_{ij}\end{matrix} / \begin{matrix}(a,3)_{i-1j-1} & (b,2)_{i-1j} \\ (a,3)_{ij-1} & (a,3)_{ij}\end{matrix}, t\right)$$
$$\left(1, \begin{matrix}(a,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (b,2)_{ij}\end{matrix} / \begin{matrix}(a,3)_{i-1j-1} & (a,3)_{i-1j} \\ (a,3)_{ij-1} & (b,2)_{ij}\end{matrix}, t\right)$$

for
- $1 \leq i, j \leq n$,
- $a, b \in \mathcal{C}$.

These rules are used to complete the set of edge pixels of our image.

15. $(t, (a,3)_{ij}/\lambda, 1)$
    for
    - $1 \leq i, j \leq n$,
    - $a \in \mathcal{C}$.

    These rules send to the cell 1 the edge pixels.

16. We can find more than one copy of an specific edge pixel, so if we wish only one copy of each edge pixel we can add a new type of rules:
    $(t, (a,3)_{ij}(a,3)_{ij}/(a,3)_{ij}, 1)$
    for
    - $1 \leq i, j \leq n$,
    - $a \in \mathcal{C}$.

- $i_\Pi = o_\Pi = 1$.

4 The Hardware Design

In [11], some preliminary segmentation results were obtained using the *tissue simulator* developed in [3]. Such a *tissue simulator* follows one of the common features of the first generation of simulators of P systems (see [13]): the lack of efficiency in favor of expressiveness. Therefore, experiments performed using this tool were extremely slow, and it could only use synthetic images of at most 30×30 pixels. Recently, a new sequential software was presented in [14], implementing ideas borrowed from [11].

In order to make the hardware design of a tissue-like P system there are several considerations that must be considered:

1. On a tissue-like P system, not only each cell evolve in a parallel manner. Every rule in every cell must be executed as many times as possible at each step. Thus, if we want that the hardware system to work exactly like the theoretical model, the system has to implement as many *minimal computation units* as the maximum number of rules in all the cells that could be executed in the same step in order to be fully parallel. If we are designing a general tissue-like P system which we want to use to configure different tissue-like P systems that solve specific problems, this is probably the main problem, as this number is defined by each P system configuration. In this case, the only way to do this is

to design the *minimal computation units* as small as possible in terms of chip area, in order to have the maximum number of them. Then, if this number is not enough to solve our problem, the system can be designed in order to do the following:

- Separate each conflicting step, into two or more sub-steps. So, in the first sub-step the system executes all the possible rules, using a piece of memory to save the results, and then it continues executing the rest of the rules that could not be executed before due to insufficient *minimal computation units* .
- Connect with other clone system(s) to solve the hole problem using more computation capacity. This options is not always possible and, in general, it is more difficult to design that the first one, but it can be the best choice dealing with hard computation problems.

On the other hand, if we want to design a specific P system, usually the best choice to deal with this issue is finding sets of rules that are mutually exclusive, that is, rules that we know that if the executing condition is true for one of them, then we know that the other ones in the same set cannot be executed. Thus, in fact we have only to design one *minimal computation unit* for each set of rules, optimizing the system area. This is the case of the described segmentation problem, in which we know that we can define only one set of rules mutually exclusive for each pixel, so in fact the system will have as many *minimal computation units* as the biggest segmentation. So the computational order will be constant, and the spatial order will be lineal.

2. The copy rules are necessary in the theoretical tissue-like P system, but in the hardware design it is not necessary in general to implement them as rules like the other ones, since they can be seen as parallel readings of some information.

   So usually those rules can be ignored in the design, seeing them as multi-lectures of the data that is trying to copy the rule. Also is easily to transform those rules into asynchronous rules. That is the segmentation case that we present, where *main rules* are synchronized by the clock system representing the synchronous P system, and the *copy rules* are asynchronous and are implicit in the interconnection circuit of the design.
3. Depending on the variant of tissue-like P systems we work with, cells could create or remove other cells during the execution in order to solve the problem. This is one of the biggest problems when simulating tissue-like P systems in software in a efficient way, but could not be the case in hardware. The FPGAs can be reconfigured while the system is still working. This feature help us to design the addition or removal cell rules as partial *on air* reconfigurations of the system *on air* easily. The only thing that we have to worry about is that those operations are not fast in terms of time, so the steps with those rules will be slower than the other ones. Because of that, trying to avoid those kind of rules while defining the P system is a good practice. The segmentation problem described has no rule of this kind.

4. The halting condition can be redefined in order to save some final computation step. This is the case of rule 9 on the segmentation problem. In most of synchronous P systems, we can know that the system has finished without make an explicit operation. For example, in our design we know that the system stops three clock cycles after the beginning. Another simple option can be found by observing the system behavior.

5. The system has to be always running while there are instances of the problem that does not have been solved yet. In fact that is a consideration that have to be done every time a hardware design is made, besides designing the P system it is important that it can return the results whereas the system is starting with a new problem. So, perhaps this is a consideration that has to be only considered not only in the design step, but in the previous theoretical P system definition before.

4.1 Segmentation Problem Design Based on FPGA

Following the segmentation example, an formal hardware system design based on the tissue-like P system described above is shown in Figures 1, 2, 3 and 4. It has been done by following the previous considerations. The system consists on *processing units* capable of dealing with 4×4 images. These units can be combined like a puzzle in order to process $n \times m$ images.

Each pixel in the image is codified with 56 bits in order to represent the theoretical objects (it contains color information, original color information, and type of object). Using this codification, a 4×4 section of the initial image is passed through the image port of each *processing unit*. Also additional information about the neighborhood of the 4×4 is required in order to work correctly, using the $blec, blrec, trec, tlec$, and bb, rb, tb, lb buses for that. If the neighborhood (or a part of it), does not exist those inputs will be at high impedance (*ghost pixel*).

The t and k ports specify the maximum distance between the pixel and its average (noise filter), and the number of different levels for the thresholding respectively. The different system steps are controlled by the clock signal (CLK).

As it is shown in figure 1, inside the *processing unit* are 16 *pixel processing units* capable of execute any rule for each pixel and each step (using the techniques described before in the first point.). The signal *change* is used to feed this units with the input data (the original image), or feed them with its own output (representing in that way the copy rules as described before in item two).

These *pixel processing units* shown in figure 2 receives a pixel and its neighborhood, and the t, k and CLK signals, and send this information to four units that implements the four sets of rules mutually exclusive mentioned before. The results are collected and processed as output. In general a fixed group of *pixel processing units* will represent a fixed group of cells in the theoretical model. But looking at the described tissue-like P, we have that each cell except cells zero and one is representing the computation of one pixel, so there is exactly one *pixel processing*

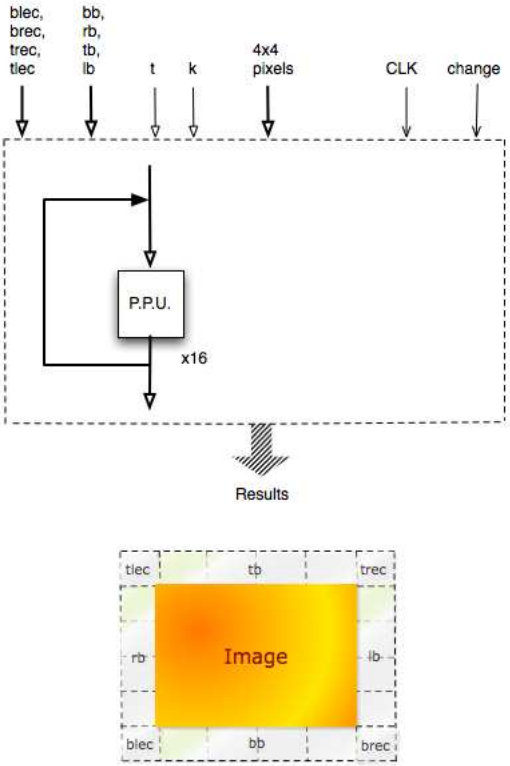


Fig. 1. Processing unit, and neighborhood of an image

unit for each cell. Then we can say that these units represents each cell in the theoretical model except zero and one.

The four units that implement the four sets of rules mutually exclusive are shown in figure 3: removing noise rules (types 3 and 4), thresholding rules (type 5), rules of the first part of the segmentation (type 9) and rules of the second part of the segmentation (type 14). The rest of rules of the theoretical family of systems are rules of coping and sending objects. These units detect automatically if the input data is a corner, an edge, or an interior pixel. Finally, in order to deal with bigger images, we can use the $blec, blrec, trec, tlec$, and bb, rb, tb, lb buses to interconnect as many as *processing units* we need (figure 4). A very simple interconnection circuit is necessary in order to give the input data to the different processing units.

The implementation of this hardware tool allows the system to apply the maximum number of rules at each moment, using the *pixel units* to solve the whole problem. Therefore, the system works exactly like the theoretical model in terms of complexity, time, concurrency and results. As said before, the implementation

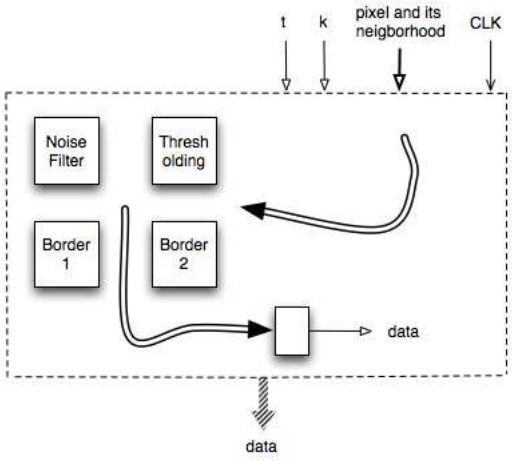


Fig. 2. Pixel Processing Unit

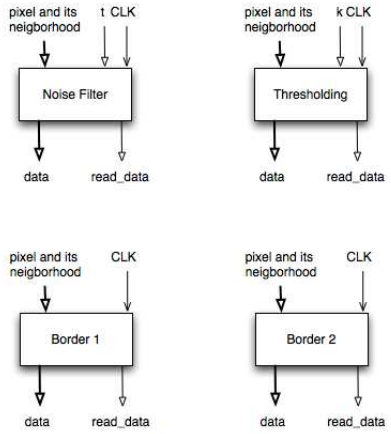


Fig. 3. Chips that implements the sets of rules mutually exclusive

of this design reveals that in fact the system is able to process any image of size $n \times m$ by using at most four clock cycles.

In figure 5, it is shown a simulation of the code following the described design that deals with 16×16 images, and some simple results using a SP605 Xilinx board with a Spartan 6 XC6SLX45T FPGA chip.

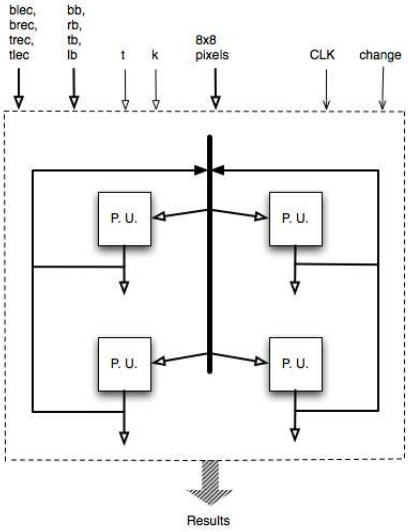


Fig. 4. Processing Units Interconnection

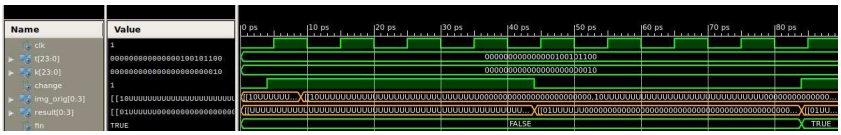


Fig. 5. Hardware design

Fig. 6. 16x16 black and white image segmentation

Fig. 7. 16x16 black and white image segmentation

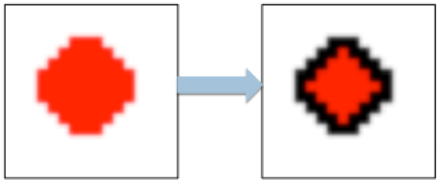


Fig. 8. 16x16 color image segmentation

5 Conclusions and Future Works

Problems associated with the treatment of Digital Images have several interesting features from a bio-inspired point of view. One of them is that they can be suitable for parallel processing, since the same sequential algorithm is usually applied in different regions of the image.

In this paper, we study the advantages and drawbacks of considering a hardware implementation of tissue-like P systems solving the segmentation problem on a hardware programming tool (FPGA). The theoretical study has been made via the language programming VHDL [2] and currently we are in the process of the real hardware implementation.

In addition, although the segmentation example showed here is a synchronous tissue-like P system, we want in the next future to work with asynchronous tissue-like P systems in order to optimize performance.

Many questions remains open as future work. One of them is the treatment of the noise in images with Membrane Computing techniques, or the parallelization and automatization of the choice of the threshold by artificial intelligence techniques.

Acknowledgements

DDP and MAGN acknowledge the support of the projects TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence of the Junta de Andalucía, grant P08-TIC-04200. JC acknowledges the support of the project MTM2009-12716 of the Ministerio español de Educación y Ciencia, the project PO6-TIC-02268 of Excellence of Junta de Andalucía, and the *Computational Topology and Applied Mathematics* PAICYT research group FQM-296.

References

1. Abdala, D.D., Jiang, X.: Fiber segmentation using constrained clustering. In: Zhang, D., Sonka, M. (eds.) ICMB. Lecture Notes in Computer Science, vol. 6165, pp. 1–10. Springer (2010)

2. Ashenden, P.J.: The Designer's Guide to VHDL. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edn. (2001)
3. Borrego-Ropero, R., Díaz-Pernil, D., Pérez-Jiménez, M.J.: Tissue simulator: A graphical tool for tissue P systems. In: Vaszil, G. (ed.) Proceedings of the International Workshop Automata for Cellular and Molecular Computing. pp. 23–34. MTA SZTAKI, Budapest, Hungary (August 2007), satellite of the 16th International Symposium on Fundamentals of Computational Theory
4. Carnero, J., Díaz-Pernil, D., Molina-Abril, H., Real, P.: Image segmentation inspired by cellular models using hardware programming. Image-A 1(3), 143–150 (2010)
5. 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 et al. [27], pp. 227–241
6. 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)
7. Cecilia, J.M., García, J.M., Guerrero, G.D., Matínez-de-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)
8. Ceterchi, R., Gramatovici, R., Jonoska, N., Subramanian, K.G.: Tissue-like P systems with active membranes for picture generation. Fundamenta Informaticae 56(4), 311–328 (2003)
9. Ceterchi, R., Mutyam, M., Păun, Gh., Subramanian, K.G.: Array-rewriting P systems. Natural Computing 2(3), 229–249 (2003)
10. Chao, J., Nakayama, J.: Cubical singular simplex model for 3D objects and fast computation of homology groups. In: 13th International Conference on Pattern Recognition (ICPR'96). vol. IV, pp. 190–194. IEEE Computer Society, IEEE Computer Society, Los Alamitos, CA, USA (1996)
11. 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.) CIARP. Lecture Notes in Computer Science, vol. 5856, pp. 169–176. Springer (2009)
12. Ciobanu, G., Wenyuan, G.: P systems running on a cluster of computers. In: Martín-Vide et al. [19], pp. 123–139
13. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Mario J. Pérez-Jiménez, M.: Software for P systems. In: Păun et al. [28], pp. 437–454
14. 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 (2010)
15. Gershoni, R., Keinan, E., Păun, Gh., Piran, R., Ratner, T., Shoshani, S.: Research topics arising from the (planned) P systems implementation experiment in Technion. In: Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Sixth Brainstorming Week on Membrane Computing. pp. 183–192. Fénix Editora, Sevilla, Spain (2008)
16. 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)

17. Kim, S.H., Kim, H.G., Tchah, K.H.: Object oriented face detection using colour transformation and range segmentation. Electronics Letters, IEEE 34, 979–980 (1998)
18. Kropatsch, W.G., Haxhimusa, Y., Ion, A.: Multiresolution image segmentations in graph pyramids. In: Kandel, A., Bunke, H., Last, M. (eds.) Applied Graph Theory in Computer Vision and Pattern Recognition, Studies in Computational Intelligence, vol. 52, pp. 3–41. Springer (2007)
19. Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers, Lecture Notes in Computer Science, vol. 2933. Springer (2004)
20. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
21. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON. Lecture Notes in Computer Science, vol. 2387, pp. 290–299. Springer (2002)
22. Nguyen, V., Kearney, D., Gioiosa, G.: Balancing performance, flexibility, and scalability in a parallel computing platform for membrane computing applications. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 385–413. Springer (2007)
23. Nguyen, V., Kearney, D., Gioiosa, G.: An algorithm for non-deterministic object distribution in p systems and its implementation in hardware. In: Corne, D.W., Frisco, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5391, pp. 325–354. Springer (2008)
24. Nguyen, V., Kearney, D., Gioiosa, G.: A region-oriented hardware implementation for membrane computing applications. In: Păun et al. [27], pp. 385–409
25. Nguyen, V., Kearney, D., Gioiosa, G.: An extensible, maintainable and elegant approach to hardware source code generation in reconfig-P. Journal of Logic and Algebraic Programming 79(6), 383–396 (2010)
26. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin, Germany (2002)
27. Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol. 5957. Springer (2010)
28. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
29. Petreska, B., Teuscher, C.: A reconfigurable hardware membrane system. In: Martín-Vide et al. [19], pp. 269–285
30. Ritter, G.X., Wilson, J.N., Davidson, J.L.: Image algebra: An overview. Computer Vision, Graphics, and Image Processing 49(3), 297–331 (1990)
31. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
32. Syropoulos, A., Mamatas, L., Allilomes, P.C., Sotiriades, K.T.: A distributed simulation of transition P systems. In: Martín-Vide et al. [19], pp. 357–368
33. Tarabalka, Y., Chanussot, J., Benediktsson, J.A.: Segmentation and classification of hyperspectral images using Watershed transformation. Pattern Recognition 43(7), 2367–2379 (2010)

34. Tobias, O.J., Seara, R.: Image segmentation by histogram thresholding using fuzzy sets. IEEE Transactions on Image Processing 11(12), 1457–1465 (2002)
35. Trimberger, S.M.: Field-Programmable Gate Array Technology. Kluwer Academic Publishers, Norwell, MA, USA (1994)
36. Wang, D., Lu, H., Zhang, J., Liang, J.Z.: A knowledge-based fuzzy clustering method with adaptation penalty for bone segmentation of ct images. In: Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference. pp. 6488–6491 (2005)
37. Yazid, H., Arof, H.: Image segmentation using watershed transformation for facial expression recognition. In: IFMBE Proceedings, 4th Kuala Lumpur International Conference on Biomedical Engineering. pp. 575–578 (2008)
38. Yuan, X., Situ, N., Zouridakis, G.: A narrow band graph partitioning method for skin lesion segmentation. Pattern Recognition 42(6), 1017–1028 (2009)
39. NVIDIA Corporation. NVIDIA CUDA^{TM} Programming Guide. `http://www.nvidia.com/object/cuda_home_new.html`
40. P system web page. http://ppage.psystems.eu

P Systems with Replicator Dynamics: A Proposal

Matteo Cavaliere[1], Miguel A. Gutiérrez-Naranjo[2]

[1] Spanish National Biotechnology Centre
(CNB-CSIC), Madrid 28049, Spain
mcavaliere@cnb.csic.es
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
magutier@us.es

Summary. This short note proposes some ideas for considering evolutionary game theory in the area of membrane computing.

1 Introduction

Evolutionary game theory is a field started by J. Maynard Smith [4] with the aim of modelling the evolution of animal behavior by using game theory. Replicator dynamics [2] is a specific type of evolutionary dynamics where individuals, called replicators, exist in several different types. Each type of individual uses a pre-programmed strategy and passes this behavior to its descendants without modification. Replicator dynamics is one of the is one of most used approach to define the evolutionary dynamics of a population.

The main idea of the mechanism the following one. One assumes a population of players (individuals/organisms) interacting in a game composed by several possible strategies. Individuals have fixed strategies. The players randomly interact with other individuals (if space is considered, then the interactions are done according to the defined structure). Each of these encounters produces a payoff for the two individuals that depend on their strategies and on the payoff matrix that defines the game. The payoff of all these encounters are added up. Payoff is interpreted as fitness (reproductive success). Strategies that do well reproduce faster. Strategies that do poorly are outcompeted.

In this note we propose the possibilities of consider replicator dynamics in the framework of Membrane Computing (P Systems), [3].

We imagine two possibilities. The first one is using replication dynamics as "evolution" rules of a membrane system. A second possibility consists in "simulating" replication dynamics by using the tools and the notions provided by the membrane computing paradigms.

We believe that both possibilities could be sources of new kinds of problems for the area.

2 Using Replicator Dynamics in P Systems

As a simple example of replication dynamics let us consider the following payoff matrix of a well-known game, the *prisoner's dilemma*, [2].

	cooperate	defect
cooperate	$4, 4$	$1, 5$
defect	$5, 1$	$2, 2$

This is read in the following way. When a cooperator meets another cooperator, they both gets 4. If a cooperator meets a defector, the cooperator gets 1 and the defector 5. If two defectors meet, they both gets 2. If we have a population of n individuals, k of them being cooperators (symbol c) and $n - k$ being defectors (symbol d) then the population is updated in the following manner (*one step of the evolutionary dynamics*).

Each object c receives a payoff that is the sum of all the payoffs obtained by considering the meetings with all other players. In this case the payoff accumulated by each single c is: $4(k - 1) + 1(n - k)$. In the same manner, each d receives $2(n - k - 1) + 5k$. The replication dynamics impose that each object (c or d) replicates (produce off-springs) with a rate function of the obtained payoff (in other words, the payoff is interpreted as fitness, [4]).

The simplest approach could then assume that each object c divides in $4(k - 1) + 1(n - k)$ copies (off-springs), while each d divides in $2(n - k - 1) + 5k$ copies. This means that each c produces $4(k-1)+1(n-k)$ copies of c and each d produces $2(n - k - 1) + 5k$ copies of d.

Moreover, one can also assume that, at each step, a certain number of objects is removed from the population. The simplest scenario is to assume a death/removal rate that indicates the number of objects (constant) removed at each step. In a more complex scenario, the removal, as the replication, could depend on the accumulated payoff (e.g., the players with worst fitness are removed). Many variants have been considered [2].

In P systems, there is the notion of compartment that has been shown to be relevant for the evolutionary dynamics of a population [2]. In this respect, there are many examples that show that the evolutionary dynamics can be very different when observed in structured populations and in homogeneous populations (e.g., [2]). One could then consider a P system where the objects in the compartments represent the individuals (players) of a population. Each object indicates (is associated to) the strategy of a certain chosen game (for instance, in the case of the prisoner's dilemma (PD), we have objects c and d).

The population (e.g, a multiset over the alphabet c and d in the PD game) evolves, in parallel, in the compartments, according to the replicator dynamics.

Specifically, the payoff matrix is used to calculate the payoff for each individual object (as described above), by considering all other objects present in the same compartment. Then, based on these obtained payoffs, one decides which objects to replicate and which objects to remove. For instance, this could be done using thresholds (e.g., *if payoff* $>$ *..then replicates*, *if payoff* $<$ *..then the object is removed*). Each object is then replicated (e.g., a c creates more copies of c, a d creates more copies of d) or is removed based on such threshold and on its obtained payoff. Target indications could be used to move the created objects across compartments. The number of objects in a certain compartment could be naturally interpreted as output produced. However, the programmability of such device remains an open issue. In fact, notice that, differently from standard P systems, the rules here cannot be programmed – they are "naturally" assigned by the evolutionary dynamics.

3 Simulating Replicator Dynamics

The second possibility is to program the replication dynamics using the tools available in the membrane computing area. The task is non-trivial, in particular to implement the payoff-based replication that is naturally present in the replicator dynamics.

We propose a first solution where any individual produces a new set of individuals identical to the original, at each time unit according to a discrete global clock. The number of off-spring depends on the number of encounters with defectors and cooperators and their corresponding payoffs.

We suggest a family of P systems for dealing with prisoner's dilemmas in its most general form (however, the approach proposed here can be generalized to different games). The family of P systems considers the following initial situation: A population of n individuals, k of them being cooperators (c) and $n-k$ being defectors (d). Let us consider four non negative integers R, S, T and P and the following general payoff matrix for the prisoner's dilemma.

	cooperate	defect
cooperate	R,R	S,T
defect	T,S	P,P

As standard in the area, [2], we use the terms R, *reward*, P, *punishment*, T, *temptation* and S, *sucker's payoff*. Hence, the 4-uple $PD \equiv \langle R, S, T, P\rangle$ can encode the game.

We assume the simplest replication mechanism where each individual c or d is substituted in the next *stage* (by using *mitosis* or whatever replication mode) by as many objects of the same type as its *payoff*. In other words, if c_n and d_n is the number of individuals of type c and d in the stage n, then

$$\begin{aligned} c_0 &= k \\ d_0 &= n - k \\ c_{n+1} &= R\,(c_n - 1) + S\,d_n \\ d_{n+1} &= T\,c_n + P\,(d_n - 1) \end{aligned}$$

In the membrane computing framework one can consider rules of type $c \to c^{\alpha}$ and $d \to d^{\beta}$. This reproduces the idea of replication of the original individuals. The drawback is, of course, than α and β depends on the number of individuals of the current configuration. This idea leads us to consider a set of rules $c \to c^{\alpha_1}$, $c \to c^{\alpha_2}$, $c \to c^{\alpha_3}$, ..., but even in case of having an *oracle* which decides the right rule in each configuration, we will need a potentially infinite amount of rules.

We propose an alternative solution that uses a P systems family (a P system for each 4-uple $\langle R, S, T, P\rangle$ in the framework of P systems with active membranes, [3], that computes $\{c_n, d_n\}_{n\in\mathbb{N}}$). The proposed systems have been checked with the SCPS simulator [1]. As usual in this P system model, each membrane can be crossed out by a unique object (at most) in each computation step. This feature will be used to control the flow of objects between regions.

Given a 4-uple $PD \equiv \langle R, S, T, P\rangle$ encoding a prisoner's dilemma, let us consider the following P system

$$\Pi_{PD} = \langle \Gamma, H, EC, \mu, w_e, w_s, R\rangle$$

where

- The alphabet of objects is $\Gamma = \{c, c_*, c_a, c_1, c_2, c_3, d, d_*, d_a, d_1, d_2, d_3, z, z_1, z_2, z_3, z_4\}$;
- $H = \{e, s\}$ is the set of labels;
- $EC = \{q0, q1, q2, q3, q4, qc, qd, qcc, qdd\}$ is the set of electrical charges;
- the membrane structure has only two membranes, the skin with label s an an elementary membrane with label e, $\mu = [\,[\,]_e^{q0}\,]_s^{q0}$;
- the initial multisets are $w_e = z$ and $w_s = \varnothing$. We also consider as *input*, the *population* of objects c^k and d^{n-k}. They will be placed in membrane e in the initial configuration.

We will also consider the following sets of rules

$$\begin{aligned} R_1 &\equiv [\,z\,]_e^{q0} \to [\,z_1]_e^{q1}\,[\,z_1\,]_e^{q2} \\ R_2 &\equiv [\,z_1 \to \lambda\,]_e^{q1} \\ R_3 &\equiv [\,z_1 \to \lambda\,]_e^{q2} \\ R_4 &\equiv [\,c]_e^{q1} \to c\,[\,]_e^{q3} \\ R_5 &\equiv [\,d]_e^{q1} \to d\,[\,]_e^{q3} \end{aligned}$$

In the initial configuration we have only one membrane e with the population of objects c and d and one extra object z. This extra object z produces the division (R_1) of the membrane. We have two copies of the population: one with charge $q1$ and the second one with charge $q2$.

The main idea is that all the objects in the membrane e with charge $q1$ will pass sequentially to membrane with charge $q2$. In this second membrane the *payoffs* will

be computed. The charges will be used as traffic-lights in order to control the flow of objects.

$$
\begin{aligned}
R_6 &\equiv c\,[\,]_e^{q2} \to [\,c_1\,]_e^{qc} \\
R_7 &\equiv d\,[\,]_e^{q2} \to [\,d_1\,]_e^{qd} \\
R_8 &\equiv [\,c\,]_e^{qc} \to z_4\,[\,]_e^{qcc} \\
R_9 &\equiv [\,d\,]_e^{qd} \to z_4\,[\,]_e^{qdd}
\end{aligned}
$$

When an object c or d arrives to the membrane with label $q2$ with R_6 or R_7, the calculation of the *payoff* starts. Since an individual does not meet itself in order to get a payoff, an object c or d is sent out of the membrane (R_8 or R_9).

$$
\begin{aligned}
R_{10} &\equiv [\,c_1 \to c_2 c_3\,]_e^{qc} \\
R_{11} &\equiv [\,d_1 \to d_2 d_3\,]_e^{qd} \\
R_{12} &\equiv [\,z_4 \to \lambda\,]_s^{q0}
\end{aligned}
$$

These rules $R_{10} - R_{12}$ are technical rules in order to adjust the proposed P system to the model of active membranes, where rules $c\,[\,]_e^{q2} \to [\,c_2 c_3\,]_e^{qc}$ or $[\,c\,]_e^{qc} \to \lambda\,[\,]_e^{qcc}$ are not allowed. The computation of the payoff is performed by the following rules:

$$
\begin{aligned}
R_{13} &\equiv [\,c \to c_*^R\,c\,]_e^{qcc} \\
R_{14} &\equiv [\,d \to c_*^S\,d\,]_e^{qcc} \\
R_{15} &\equiv [\,c \to d_*^T\,c\,]_e^{qdd} \\
R_{16} &\equiv [\,d \to d_*^P\,d\,]_e^{qdd}
\end{aligned}
$$

The charge qcc can be interpreted as the visit of an individual c. The objects c in the membrane produce R copies of c_* and all the objects d produce S copies of d_*. Analogously, the charge qdd can be interpreted as the visit of an individual d. In this case, the objects c in the membrane produce T copies of c_* and all the objects d produce P copies of d_*.

The path to complete the cycle and to start again begins with the following rules. An object z_2 is sent to the first membrane labeled with e in order to get a new individual for the calculation of the payoff.

$$
\begin{aligned}
R_{17} &\equiv [\,c_2\,]_e^{qcc} \to z_2\,[\,]_e^{q2} \\
R_{18} &\equiv [\,d_2\,]_e^{qdd} \to z_2\,[\,]_e^{q2} \\
R_{19} &\equiv z_2\,[\,]_e^{q3} \to [\,z_2\,]_e^{q1}
\end{aligned}
$$

The object c or d sent out by the rule R_8 or R_9 is placed again on the corresponding membrane by rule R_{20} or R_{21}.

$$
\begin{aligned}
R_{20} &\equiv [\,c_3 \to c\,]_e^{q2} \\
R_{21} &\equiv [\,d_3 \to d\,]_e^{q2}
\end{aligned}
$$

Sending z_2 into the corresponding membrane opens the traffic light by changing the charge to q_1. The cycle starts again and rules R_4 and R_5 can be triggered again, if any object c or d remains in the membrane. In order to control the behavior of the membrane when all the objects c and d have been sent out, we add some technical rules.

$R_{22} \equiv [\, z_2 \rightarrow z_3 \,]_e^{q1}$
$R_{23} \equiv [\, z_3 \rightarrow \lambda \,]_e^{q3}$

If z_3 appears in a membrane, it means that all objects c or d have been sent out in previous steps. In this case, the membrane can be dissolved and the cycle of computing the payoffs is completed.

$R_{24} \equiv [\, z_3 \,]_e^{q1} \rightarrow z_3$
$R_{25} \equiv z_3 \, [\,]_e^{q2} \rightarrow [\, z_3 \,]_e^{q4}$

When an object z_3 goes into the membrane with label e, the *old* objects c and d are removed and the objects c_* and d_* become the new population.

$R_{26} \equiv [\, c_* \rightarrow c_a \,]_e^{q4}$
$R_{27} \equiv [\, d_* \rightarrow d_a \,]_e^{q4}$
$R_{28} \equiv [\, z_3 \rightarrow z \, z_4 \,]_e^{q4}$
$R_{29} \equiv [\, c_a \rightarrow c \,]_e^{q4}$
$R_{30} \equiv [\, d_a \rightarrow d \,]_e^{q4}$
$R_{31} \equiv [\, c \rightarrow \lambda \,]_e^{q4}$
$R_{32} \equiv [\, d \rightarrow \lambda \,]_e^{q4}$

Finally, we change the charge of the membrane e and a new *stage* can start

$R_{33} \equiv [\, z_4 \,]_e^{q4} \rightarrow z_4 \, [\,]_e^{q0}$

3.1 Overview of the Computation

The main idea of the design is to consider two copies of the population. The first copy (which acts as a counter) sends individuals to the second one: when all the objects have been sent, the computation of all payoffs is completed and we finish a *cycle*. In the second copy, the payoffs are computed and stored. For each object, the P system takes five computational steps in order to calculate its payoff.

We start with the initial configuration $C_0 = [\, [\, z c^k d^{n-k} \,]_e^{q0} \,]_s^{q0}$. Initially, the two copies of the population are created by applying the rule R_1, $C_1 = [\, [\, z_1 c^k d^{n-k} \,]_e^{q1} \, [\, z_1 c^k d^{n-k} \,]_e^{q2}]_s^{q0}$. The first new membrane, with label $q1$ will send objects to the second one with label $q2$. At this moment, rules R_4 and R_5 can be non-deterministically applied, but, due to the semantics of active membranes, only one of them is chosen. Let us suppose that R_3 is taken (the other case is analogous) and we reach $C_2 = [\, [\, c^{k-1} d^{n-k} \,]_e^{q3} \, [\, c^k d^{n-k} \,]_e^{q2} \, c]_s^{q0}$. Notice that the label in

the first membrane has been changed to $q3$. Intuitively, this membrane is closed till the arrival of the object z_2 at step 6. Objects z_1 are removed.

In the next step, the object c in the skin is sent as c_1 into the second elementary membrane and changes the polarization, $C_3 = [\,[\,c^{k-1}d^{n-k}\,]_e^{q3}\,[\,c^k d^{n-k}c_1\,]_e^{qc}]_s^{q0}$. The process of computing the payoff of this object c_1 starts: c_1 is replaced by c_2c_3 and one object c is sent to the skin, changing again the polarization, $C_4 = [\,[\,c^{k-1}d^{n-k}\,]_e^{q3}\,[\,c^{k-1}d^{n-k}c_2c_3\,]_e^{qcc}\,z_4\,]_s^{q0}$. The computation of the payoff is made now by application in parallel of the rules R_{13} and R_{14}. In order to avoid that this rule can be applied in the next step, the object c_2 is sent out (as z_2) and the polarization changes again. According to R_{13} and R_{14}, R objects c_* are produced for each c and S objects c_* for each d, $C_5 = [\,[\,c^{k-1}d^{n-k}\,]_e^{q3}\,[\,c^{k-1}d^{n-k}c_3c_*^{R(k-1)+S(n-k)}\,]_e^{q2}\,z_2]_s^{q0}$. Finally, c_3 goes to c in the second elementary membrane and z_2 goes into the first one, changes the polarization and *opens* the membrane, $C_6 = [\,[\,c^{k-1}d^{n-k}z_2\,]_e^{q1}\,[\,c^k d^{n-k}c_*^{R(k-1)+S(n-k)}\,]_e^{q2}\,z_2]_s^{q0}$. Notice that we have again two membranes, one of them with charge $q1$ and the other one with charge $q2$ as in the configuration C_1. In the next steps the process goes on by sending all the objects from the first membrane to the second one, where the payoffs will be stored.

After $5n+1$ steps we arrive at the configuration

$$C_{5n+1} = [\,[\,z_2\,]_e^{q1}\,[\,c^k d^{n-k}c_*^{k(R(k-1)+S(n-k))}d_*^{(n-k)(Tk+P(n-k-1))}\,]_e^{q2}\,]_s^{q0}$$

No more objects are left in the first membrane. In C_{5n+2} we have an object z_3 inside a membrane with label e and charge q_1. In the next step, the rule r_{24} is applied and the membrane is dissolved,

$$C_{5n+3} = [\,[\,c^k d^{n-k}c_*^{k(R(k-1)+S(n-k))}d_*^{(n-k)(Tk+P(n-k-1))}\,]_e^{q2}\,z_3]_s^{q0}$$

The object z_3 goes into the elementary membrane and changes the polarization to q_4,

$$C_{5n+4} = [\,[\,c^k d^{n-k}c_*^{k(R(k-1)+S(n-k))}d_*^{(n-k)(Tk+P(n-k-1))}z_3\,]_e^{q4}\,]_s^{q0}$$

This new polarization produces the updating of the payoff to a new population in two steps, so

$$C_{5n+6} = [\,[\,c^{k(R(k-1)+S(n-k))}d^{(n-k)(Tk+P(n-k-1))}\,]_e^{q4}\,z_4\,]_s^{q0}$$

which is analogous to the configuration C_0, so a new stage starts (the object z_4 disappears in the next step by using the rule R_{12}).

4 Conclusions and Future Work

Replicator dynamics in one of most used mechanisms in evolutionary game theory. In this context, several papers have shown the relevance of compartments and structures. On the other hand, membrane computing explicitly investigates the

relevances of compartments for computation. A natural possibility, proposed in this note, is to bridge these two areas. We have sketched two possibilities. The first one consists in using rules inspired from the replicator dynamics. A second one consists in programming the replicator dynamics using the tools of membrane computing. In this case, we have presented a possible solution using membrane systems with active membranes. However several other solutions can be imagined, in particular replacing the cell-like membrane structure by a tissue-like structure could allow a simpler version of the simulations.

Acknowledgments

MAGN acknowledges the support of the projects TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence of the Junta de Andalucía, grant P08-TIC-04200. M. C. acknowledges the support of the program JAEDoc15 ("Programa junta para la ampliacion de estudios") and of the Research Group on Natural Computing of the University of Sevilla.

References

1. Gutiérrez-Naranjo, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: A simulator for confluent P systems. In: Gutiérrez-Naranjo, M.A., Riscos-Núñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Third Brainstorming Week on Membrane Computing. pp. 169–184. Fénix Editora, Sevilla, Spain (2005)
2. Hofbauer, J., Sigmund, K.: *Evolutionary Games and Population Dynamics*. Cambridge University Press (Jun 1998)
3. Păun Gh., Rozenberg G., Salomaa A. Eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
4. Smith, J.M.: *Evolution and the Theory of Games*. Cambridge University Press, 1st edition edn. (Dec 1982)

P Colonies of Capacity One and Modularity

Luděk Cienciala, Lucie Ciencialová, and Miroslav Langer

Institute of Computer Science, Silesian University in Opava, Czech Republic
{ludek.cienciala, lucie.ciencialova, miroslav.langer}@fpf.slu.cz

Summary. We continue the investigation of P colonies introduced in [8], a class of abstract computing devices composed of independent agents, acting and evolving in a shared environment. The first part is devoted to the P colonies of the capacity one. We present improved allready presented results concerning the computional power of the P colonies. The second part is devoted to the modularity of the P colonies. We deal with dividing the agents into modules.

1 Introduction

P colonies were introduced in the paper [7] as formal models of a computing device inspired by membrane systems and formal grammars called colonies. This model is inspired by structure and functioning of a community of living organisms in a shared environment.

The independent organisms living in a P colony are called agents or cells. Each agent is represented by a collection of objects embedded in a membrane.

The number of objects inside each agent is the same and constant during computation.

The environment contains several copies of the basic environmental object denoted by e. The number of the copies of e in the environment is sufficient.

With each agent a set of programs is associated. The program, which determines the activity of the agent, is very simple and depends on content of the agent and on multiset of objects placed in the environment. Agent can change content of the environment by programs and through the environment it can affect the behavior of other agents.

This influence between agents is a key factor in functioning of the P colony. In each moment each object inside the agent is affected by executing the program.

For more information about P systems see [12] or [13].

2 Definitions

Throughout the paper we assume that the reader is familiar with the basics of the formal language theory.

We use NRE to denote the family of the recursively enumerable sets of natural numbers. Let Σ be the alphabet. Let Σ^* be the set of all words over Σ (including the empty word ε). We denote the length of the word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$.

A multiset of objects M is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : V \to N$; f assigns to each object in V its multiplicity in M. The set of all multisets with the set of objects V is denoted by V°. The set V' is called the support of M and is denoted by $supp(M)$ if for all $x \in V'$ $f(x) \neq 0$ holds. The cardinality of M, denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Each multiset of objects M with the set of objects $V' = \{a_1, \ldots a_n\}$ can be represented as a string w over alphabet V', where $|w|_{a_i} = f(a_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters represent the same multiset M. The ε represents the empty multiset.

2.1 P colonies

We briefly recall the notion of P colonies. A P colony consists of agents and an environment. Both the agents and the environment contain objects. With each agent a set of programs is associated. There are two types of rules in the programs. The first type of rules, called the evolution rules, are of the form $a \to b$. It means that the object a inside the agent is rewritten (evolved) to the object b. The second type of rules, called the communication rules, are of the form $c \leftrightarrow d$. When the comunication rule is performed, the object c inside the agent and the object d outside the agent swap their places. Thus after execution of the rule, the object d appears inside the agent and the object c is placed outside the agent.

In [7] the set of programs was extended by the checking rules. These rules give an opportunity to the agents to opt between two possibilities. The rules are in the form r_1/r_2. If the checking rule is performed, then the rule r_1 has higher priority to be executed over the rule r_2. It means that the agent checks whether the rule r_1 is applicable. If the rule can be executed, then the agent is compulsory to use it. If the rule r_1 cannot be applied, then the agent uses the rule r_2.

Definition 1. *The P colony of the capacity k is a construct*
$$\Pi = (A, e, f, V_E, B_1, \ldots, B_n), \text{ where}$$

- *A is an alphabet of the colony, its elements are called objects,*
- *$e \in A$ is the basic object of the colony,*
- *$f \in A$ is the final object of the colony,*
- *V_E is a multiset over $A - \{e\}$,*
- *B_i, $1 \leq i \leq n$, are agents, each agent is a construct $B_i = (O_i, P_i)$, where*

- *O_i is a multiset over A, it determines the initial state (content) of the agent, $|O_i| = k$,*
- *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite multiset of programs, where each program contains exactly k rules, which are in one of the following forms each:*
  - *$a \to b$, called the evolution rule,*
  - *$c \leftrightarrow d$, called the communication rule,*
  - *r_1/r_2, called the checking rule; r_1, r_2 are the evolution rules or the communication rules.*

An initial configuration of the P colony is an $(n+1)$-tuple of strings of objects present in the P colony at the beginning of the computation. It is given by the multiset O_i for $1 \leq i \leq n$ and by the set V_E. Formally, the configuration of the P colony Π is given by $(w_1, \ldots, w_n, w_E)$, where $|w_i| = k,\ 1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent, and $w_E \in (A - \{e\})^\circ$ represents all the objects in the environment different from the object e.

In the paper parallel model of P colonies will be studied. At each step of the parallel computation each agent tries to find one usable program. If the number of applicable programs are higher than one, then the agent chooses one of the rule nondeterministically. At one step of the computation the maximal possible number of agents are active.

Let the programs of each P_i be labeled in a one-to-one manner by labels in a set $lab(P_i)$ in such a way that $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$, $1 \leq i, j \leq n$.

To express derivation step formally, we introduce following four functions for the agent using the rule r of program $p \in P$ with objects w in the environment:

For the rule r which is $a \to b, c \leftrightarrow d$ and $c \leftrightarrow d/c' \leftrightarrow d'$ respectively, and for multiset $w \in V^\circ$ we define:

$$
\begin{array}{ll}
left\,(a \to b, w) = a & left\,(c \leftrightarrow d, w) = \varepsilon \\
right\,(a \to b, w) = b & right\,(c \leftrightarrow d, w) = \varepsilon \\
export\,(a \to b, w) = \varepsilon & export\,(c \leftrightarrow d, w) = c \\
import\,(a \to b, w) = \varepsilon & import\,(c \leftrightarrow d, w) = d
\end{array}
$$

$$
\left.
\begin{array}{l}
left\,(c \leftrightarrow d/c' \leftrightarrow d', w) = \varepsilon \\
right\,(c \leftrightarrow d/c' \leftrightarrow d', w) = \varepsilon \\
export\,(c \leftrightarrow d/c' \leftrightarrow d', w) = c \\
import\,(c \leftrightarrow d/c' \leftrightarrow d', w) = d
\end{array}
\right\} \text{for } |w|_d \geq 1
$$

$$
\left.
\begin{array}{l}
export\,(c \leftrightarrow d/c' \leftrightarrow d', w) = c' \\
import\,(c \leftrightarrow d/c' \leftrightarrow d', w) = d'
\end{array}
\right\} \text{for } |w|_d = 0 \text{ and } |w|_{d'} \geq 1
$$

For a program p and any $\alpha \in \{left, right, export, import\}$, let be

$$\alpha\,(p, w) = \cup_{r \in p} \alpha\,(r, w).$$

A transition from a configuration to another is denoted as

$$(w_1, \ldots, w_n; w_E) \Rightarrow (w'_1, \ldots, w'_n; w'_E),$$ where the following conditions are satisfied:

- There is a set of program labels P with $|P| \leq n$ such that
  - $p, p' \in P$, $p \neq p'$, $p \in lab(P_j)$ implies $p' \notin lab(P_j)$,
  - for each $p \in P$, $p \in lab(P_j)$, $left(p, w_E) \cup export(p, w_E) = w_j$, and $\bigcup_{p \in P} import(p, w_E) \subseteq w_E$.
- Furthermore, the chosen set P is maximal, i.e. if any other program $r \in \cup_{1 \leq i \leq n} lab(P_i)$, $r \notin P$ is added to P, then the conditions listed above are not satisfied.

Now, for each j, $1 \leq j \leq n$, for which there exists a $p \in P$ with $p \in lab(P_j)$, let be $w'_j = right(p, w_E) \cup import(p, w_E)$. If there is no $p \in P$ with $p \in lab(P_j)$ for some j, $1 \leq j \leq n$, then let be $w'_j = w_j$ and moreover, let be

$$w'_E = w_E - \bigcup_{p \in P} import(p, w_E) \cup \bigcup_{p \in P} export(p, w_E).$$

A configuration is halting if the set of program labels P satisfying the conditions above cannot vary from the empty set. A set of all possible halting configurations is denoted by H. A halting computation can be associated with the result of the computation. It is given by the number of copies of the special symbol f present in the environment. The set of numbers computed by a P colony Π is defined as

$$N(\Pi) = \left\{ |v_E|_f \mid (w_1, \ldots, w_n, V_E) \Rightarrow^* (v_1, \ldots, v_n, v_E) \in H \right\},$$

where $(w_1, \ldots, w_n, V_E)$ is the initial configuration, $(v_1, \ldots, v_n, v_E)$ is a halting configuration, and $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

Consider a P colony $\Pi = (A,\ e,\ f,\ V_E,\ B_1,\ \ldots,\ B_n)$. The maximal number of programs associated with the agents in the P colony Π are called the height of the P colony Π. The degree of the P colony Π is the number of agents in it. The third parameter characterizing the P colony is the capacity of the P colony Π describing the number of the objects inside each agent.

Let us use the following notations:
$NPCOL_{par}(k, n, h)$ for the family of all sets of numbers computed by the P colonies working in a parallel, using no checking rules and with:
- the capacity at most k,
- the degree at most n and
- the height at most h.

If we allow the checking rules, then the family of all sets of numbers computed by the P colonies is denoted by $NPCOL_{par}K$. If the P colonies are restricted, we use the notation $NPCOL_{par}R$, respectively $NPCOL_{par}KR$.

2.2 Register machines

The aim of the paper is to characterize the size of the families $NPCOL_{par}(k, n, h)$ comparing them with the recursively enumerable sets of numbers. To meet the target, we use the notion of a register machine.

Definition 2. *[9] A register machine is the construct* $M = (m, H, l_0, l_h, P)$ *where:*
- m *is the number of registers,*
- H *is the set of instruction labels,*
- l_0 *is the start label,* l_h *is the final label,*
- P *is a finite set of instructions injectively labeled with the elements from the set* H*.*

The instructions of the register machine are of the following forms:

$l_1 : (ADD(r), l_2, l_3)$ Add 1 to the content of the register r and proceed to the instruction (labeled with) l_2 or l_3.

$l_1 : (SUB(r), l_2, l_3)$ If the register r stores the value different from zero, then subtract 1 from its content and go to instruction l_2, otherwise proceed to instruction l_3.

$l_h : HALT$ Stop the machine. The final label l_h is only assigned to this instruction.

Without loss of generality, it can be assumed that in each ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ and in each conditional SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, the labels l_1, l_2, l_3 are mutually distinct.

The register machine M computes a set $N(M)$ of numbers in the following way: the computation starts with all registers empty (hence storing the number zero) and with the instruction labeled l_0. The computation proceeds by applying the instructions indicated by the labels (and the content of registers allows its application). If it reaches the halt instruction, then the number stored at that time in the register 1 is said to be computed by M and hence it is introduced in $N(M)$. (Because of the nondeterminism in choosing the continuation of the computation in the case of ADD-instructions, $N(M)$ can be an infinite set.) It is known (see e.g.[9]) that in this way we can compute all sets of numbers which are Turing computable.

Moreover, we call a register machine partially blind [6] if we interpret a subtract instruction in the following way: $l_1 : (SUB(r); l_2; l_3)$ - if there is a value different from zero in the register r, then subtract one from its contents and go to instruction l_2 or to instruction l_3; if there is stored zero in the register r when attempting to decrement the register r, then the program ends without yielding a result.

When the register machine reaches the final state, the result obtained in the first register is only taken into account if the remaining registers store value zero. The family of sets of non-negative integers generated by partially blind register machines is denoted by NRM_{pb}. The partially blind register machine accepts a proper subset of NRE.

3 P colonies with one object inside the agent

In this Section we analyze the behavior of P colonies with only one object inside each agent of P colonies. It means that each program is formed by only one rule,

either the evolution rule or the communication rule. If all agents have their programs with the evolution rules, the agents "live only for themselves" and do not communicate with the environment.

Following results were proved:
- $NPCOL_{par}K(1,*,7) = NRE$ in [1],
- $NPCOL_{par}K(1,4,*) = NRE$ in [2],
- $NPCOL_{par}(1,2,*) = NPBRM$ in [2].

Theorem 1. $NPCOL_{par}(1,4,*) = NRE$

Proof. We construct a P colony simulating the computation of the register machine. Because there are only copies of e in the environment and inside the agents, we have to initialize a computation by generating the initial label l_0. After generating the symbol l_0, the agent stops. It can continue its activity only by using a program with the communication rule. Two agents will cooperate in order to simulate the ADD and SUB instructions. Let us consider an m-register machine $M = (m, H, l_0, l_h, P)$ and present the content of the register i by the number of copies of a specific object a_i in the environment. We construct the P colony

$\Pi = (A, e, f, \emptyset, B_1, \ldots, B_4)$ with:
- alphabet $A = \{l_i, L_i, \textcircled{l_i}, \boxed{l_i}, \boxed{L_i}, m_i, m_i', \textcircled{m_i}, \boxed{m_i}, y_i, n_i, \mid 0 \leq i \leq |H|\} \cup$
$\cup \{a_i | 1 \leq i \leq m\} \cup \{A_r^i \mid \text{for every } l_i : (SUB(r), l_j, l_k) \in H\} \cup$
$\cup \{e, d, C\}$,
- $f = a_1$,
- $B_i = (e, P_i),\ 1 \leq i \leq 4$.

(1) To initialize simulation of computation of M we define the agent $B_1 = (e, P_1)$ with a set of programs:

P_1 :
$1 : \langle e \rightarrow l_0 \rangle ;$

(2) We need an additional agent to generate a special object d. This agent will be working during whole computation. In each pair of steps the agent B_2 places a copy of d to the environment. This agent stops working when it consumes the symbol which is generated by the simulation of the instruction l_h from the environment.

P_2 :
$2 : \langle e \rightarrow d \rangle , 3 : \langle d \leftrightarrow e \rangle , 4 : \langle d \leftrightarrow l_h \rangle ;$

The P colony Π starts its computation in the initial configuration $(e, e, e, e, \varepsilon)$. In the first subsequence of steps of the P colony Π only agents B_1, B_2 can apply their programs.

	configuration of Π								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	e	e	e	e		1	2		
2.	l_0	d	e	e			3		
3.	l_0	e	e	e	d		2		

(3) To simulate the ADD-instruction $l_1 : (ADD(r), l_2, l_3)$, we define two agents B_3 and B_4 in the P colony Π. These agents help each other to add a copy of the object a_r and the object l_2 or l_3 into the environment.

P_1	P_1	P_3	P_3
$5 : \langle l_1 \leftrightarrow D_1 \rangle$,	$9 : \left\langle \boxed{l_1} \rightarrow \boxed{L_1} \right\rangle$,	$13 : \langle e \leftrightarrow D_1 \rangle$,	$16 : \langle e \leftrightarrow \textcircled{l_1} \rangle$,
$6 : \langle D_1 \leftrightarrow d \rangle$,	$10 : \left\langle \boxed{L_1} \rightarrow L_1 \right\rangle$,	$14 : \left\langle D_1 \rightarrow \boxed{l_1} \right\rangle$,	$17 : \langle \textcircled{l_1} \rightarrow a_r \rangle$,
$7 : \langle d \rightarrow \textcircled{l_1} \rangle$,	$11 : \langle L_1 \rightarrow l_2 \rangle$,	$15 : \left\langle \boxed{l_1} \leftrightarrow e \right\rangle$,	$18 : \langle a_r \leftrightarrow e \rangle$,
$8 : \left\langle \textcircled{l_1} \leftrightarrow \boxed{l_1} \right\rangle$,	$12 : \langle L_1 \rightarrow l_3 \rangle$,		

This pair of agents generate two objects. One object increments value of the particular register and the second one defines of which instruction will simulation continue. One agent is not able to generate both objects corresponding to the simulation of one instruction, because at the moment of placing all of its content into the environment via the communication rules, it does not know which instruction it simulates. It nondeterministically chooses one of the possible instructions. Now it is necessary to check whether the agent has chosen the right instruction. For this purpose the second agent slightly changes first generated object. The first agent swaps this changed object for the new one generated only if it belongs to the same instruction. If this is not done succesfully, the computation never stops because of absence of the halting object for the agent B_2.

An instruction $l_1 : (ADD(r), l_2, l_3)$ is simulated by the following sequence of steps. Let the content of the agent B_2 be d.

	configuration of Π								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	l_1	d	e	e	$a_r^u d^v$	5	3		
2.	D_1	e	e	e	$a_r^u d^{v+1}$	6	2		
3.	d	d	e	e	$D_1 a_r^u d^v$	7	3	13	
4.	$\textcircled{l_1}$	e	D_1	e	$a_r^u d^{v+1}$		2	14	
5.	$\textcircled{l_1}$	d	$\boxed{l_1}$	e	$a_r^u d^{v+1}$		3	15	
6.	$\textcircled{l_1}$	e	e	e	$\boxed{l_1} a_r^u d^{v+2}$	8	2		
7.	$\boxed{l_1}$	d	e	e	$\textcircled{l_1} a_r^u d^{v+2}$	9	3	16	
8.	$\boxed{L_1}$	e	$\textcircled{l_1}$	e	$a_r^u d^{v+3}$	10	2	17	
9.	L_1	d	a_r	e	$a_r^u d^{v+3}$	**11** or 12	3	18	
10.	l_2	e	e	e	$a_r^{u+1} d^{v+4}$				

(4) For each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, the following programs are introduced in the sets P_1, P_3 and in the set P_4:

P_1	P_1	P_3	P_3
$19 : \langle l_1 \to D_1 \rangle$,	$28 : \langle l_1' \to n^1 \rangle$,	$36 : \langle e \leftrightarrow D_1 \rangle$,	$45 : \langle e \leftrightarrow y^1 \rangle$,
$20 : \langle D_1 \leftrightarrow d \rangle$,	$29 : \langle n^1 \leftrightarrow m_1 \rangle$,	$37 : \langle D_1 \to \boxed{l_1} \rangle$,	$46 : \langle y^1 \to l_2 \rangle$,
$21 : \langle d \to (l_1) \rangle$,	$30 : \langle m_1 \to m_1' \rangle$,	$38 : \langle \boxed{l_1} \leftrightarrow e \rangle$,	$47 : \langle l_2 \leftrightarrow e \rangle$,
$22 : \langle (l_1) \leftrightarrow \boxed{l_1} \rangle$,	$31 : \langle m_1' \to m_1'' \rangle$,	$39 : \langle e \leftrightarrow (l_1) \rangle$,	$48 : \langle e \leftrightarrow n^1 \rangle$,
$23 : \langle \boxed{l_1} \to A_r^1 \rangle$,	$32 : \langle m_1'' \to (m_1) \rangle$,	$40 : \langle (l_1) \to l_1' \rangle$,	$49 : \langle n_1 \to l_3 \rangle$,
$24 : \langle A_r^1 \leftrightarrow a_r \rangle$,	$33 : \langle (m_1) \leftrightarrow l_2 \rangle$,	$41 : \langle l_1' \leftrightarrow e \rangle$,	$50 : \langle l_3 \leftrightarrow e \rangle$,
$25 : \langle A_r^1 \leftrightarrow l_1' \rangle$,	$34 : \langle (m_1) \to \boxed{m_1} \rangle$,	$42 : \langle e \leftrightarrow A_r^1 \rangle$,	
$26 : \langle a_r \to y^1 \rangle$,	$35 : \langle \boxed{m_1} \leftrightarrow l_3 \rangle$,	$43 : \langle A_r^1 \to m_1 \rangle$,	
$27 : \langle y^1 \leftrightarrow m_1 \rangle$,		$44 : \langle m_1 \leftrightarrow e \rangle$,	

Agents B_1, B_3 and B_4 collectively check the state of particular register and generate label of following instruction. Part of the simulation is devoted to purify the environment from redundant objects.

P_4	
$51 : \langle e \leftrightarrow (m_1) \rangle$,	$54 : \langle l_1' \to e \rangle$,
$52 : \langle (m_1) \to d \rangle$,	$55 : \langle e \leftrightarrow \boxed{m_1} \rangle$,
$53 : \langle d \leftrightarrow l_1' \rangle$,	$56 : \langle \boxed{m_1} \to e \rangle$,

The instruction $l_1 : (SUB(r), l_2, l_3)$ is simulated by the following sequence of steps. If the value in counter r is zero:

	configuration of Π					applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	l_1	d	e	e	d^v	19	3		
2.	D_1	e	e	e	d^{v+1}	20	2		
3.	d	d	e	e	$D_1 d^v$	21	3	36	
4.	(l_1)	e	D_1	e	d^{v+1}		2	37	
5.	(l_1)	d	$[l_1]$	e	d^{v+1}		3	38	
6.	(l_1)	e	e	e	$[l_1] d^{v+2}$	22	2		
7.	$[l_1]$	d	e	e	$(l_1) d^{v+2}$	23	3	39	
8.	A_r^1	e	(l_1)	e	d^{v+3}		2	40	
9.	A_r^1	d	l_1'	e	d^{v+3}		3	41	
10.	A_r^1	e	e	e	$l_1' d^{v+4}$	25	2		
11.	l_1'	d	e	e	$A_r^1 d^{v+4}$	28	3	42	
12.	n_1	e	A_r^1	e	d^{v+5}		2	43	
13.	n_1	d	m_1	e	d^{v+5}		3	44	
14.	n_1	e	e	e	$m_1 d^{v+6}$	29	2		
15.	m_1	d	e	e	$n_1 d^{v+6}$	30	3	48	
16.	m_1'	e	n_1	e	d^{v+7}	31	2	49	
17.	m_1''	d	l_3	e	d^{v+7}	32	3	50	
18.	(m_1)	e	e	e	$l_3 d^{v+8}$	34	2		
19.	$[m_1]$	d	e	e	$l_3 d^{v+8}$	35	3		
20.	l_3	e	e	e	$[m_1] d^{v+9}$		2		55
21.	l_3	d	e	$[m_1]$	d^{v+9}		3		56
22.	l_3	e	e	e	d^{v+10}		2		

If the register r stores a value different from zero:

	configuration of Π					applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	l_1	d	e	e	$a_r^u d^v$	19	3		
2.	D_1	e	e	e	$a_r^u d^{v+1}$	20	2		
3.	d	d	e	e	$D_1 a_r^u d^v$	21	3	36	
4.	(l_1)	e	D_1	e	$a_r^u d^{v+1}$		2	37	
5.	(l_1)	d	$[l_1]$	e	$a_r^u d^{v+1}$		3	38	
6.	(l_1)	e	e	e	$[l_1] a_r^u d^{v+2}$	22	2		
7.	$[l_1]$	d	e	e	$(l_1) a_r^u d^{v+2}$	23	3	39	

	configuration of Π					applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
8.	A_r^1	e	(1)	e	$a_r^u d^{v+3}$	24	2	40	
9.	a^r	d	l_1'	e	$A_r^1 a_r^{u-1} d^{v+3}$	26	3	41	
10.	y_1	e	e	e	$l_1' A_r^1 a_r^{u-1} d^{v+4}$		2	42	
11.	y_1	d	A_r^1	e	$l_1' a_r^{u-1} d^{v+4}$	28	3	43	
12.	y_1	e	m_1	e	$l_1' a_r^{u-1} d^{v+5}$		2	44	
13.	y_1	d	e	e	$m_1 l_1' a_r^{u-1} d^{v+5}$	27	3		
14.	m_1	e	e	e	$y_1 l_1' a_r^{u-1} d^{v+6}$	30	2	45	
15.	m_1'	d	y_1	e	$l_1' a_r^{u-1} d^{v+6}$	31	3	46	
16.	m_1''	e	l_2	e	$l_1' a_r^{u-1} d^{v+7}$	32	2	47	
17.	(m_1)	d	e	e	$l_2 l_1' a_r^{u-1} d^{v+7}$	33	3		
18.	l_2	e	e	e	(m_1)$l_1' a_r^{u-1} d^{v+8}$		2		51
19.	l_2	d	e	(m_1)	$l_1' a_r^{u-1} d^{v+8}$		3		52
20.	l_2	e	e	d	$l_1' a_r^{u-1} d^{v+9}$		2		53
21.	l_2	d	e	l_1'	$a_r^{u-1} d^{v+10}$		3		54
22.	l_2	e	e	e	$a_r^{u-1} d^{v+11}$		2		

(5) The halting instruction l_h is simulated by the agent B_1 with a subset of programs:

$$\frac{P_1}{57 : \langle l_h \leftrightarrow d \rangle}.$$

The agent places the object l_h into the environment, from where it can be consumed by the agent B_2 and by this the agent B_2 stops its activity.

	configuration of Π								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	l_h	e	e	e	d^v	57	2		
2.	d	e	d	e	$l_h d^v$		4		
3.	d	l_h	e	e	d^{v+1}				

The P colony Π correctly simulates computation of the register machine M. The computation of Π starts with no object a_r placed in the environment in the same way as the computation of M starts with zeros in all registers. The computation of Π stops if the symbol l_h is placed inside the agent B_2 in the same way as M stops by executing the halting instruction labeled l_h. Consequently, $N(M) = N(\Pi)$ and because the number of agents equals four, the proof is complete. □

Theorem 2. $NPCOL_{par}(1, *, 8) = NRE$

Proof. We construct a P colony simulating the computation of the register machine. Because there are only copies of e in the environment and inside the agents, we have to initialize a computation by generating the initial label l_0. After generating the symbol l_0 this agent stops and it can start its activity only by using

a program with the communication rule. Two agents will cooperate in order to simulate the ADD and SUB instructions.

Let us consider an m-register machine $M = (m, H, l_0, l_h, P)$ and present the content of the register i by the number of copies of a specific object a_i in the environment. We construct the P colony

$\Pi = (A, e, f, \emptyset, B_1, \ldots, B_n),\ n = |H| + 2$ where:

– alphabet $A = \{l_i, l_i', i', i'', \textcircled{i}, \boxed{i}, D_i, \overline{l_i} | 0 \leq i \leq |H|\} \cup$
$\cup\ \{a_i | 1 \leq i \leq m\} \cup \{e, d\}$,

– $f = a_1$,

– $B_i = (e, P_i),\ 1 \leq i \leq 4$.

(1) To initialize simulation of computation of M, we define the agent $B_1 = (e, P_1)$ with a set of programs:

P_1 :

$1 : \langle e \rightarrow l_0 \rangle, 2 : \langle l_0 \leftrightarrow d \rangle;$

(2) We need an additional agent to generate a special object d. This agent will be working during whole computation. In each pair of steps the agent B_2 places a copy of d to the environment..

P_2 :

$3 : \langle e \rightarrow d \rangle, 4 : \langle d \leftrightarrow e \rangle, 5 : \langle d \leftrightarrow l_h \rangle;$

The P colony Π starts its computation in the initial configuration $(e, e, e, e, \varepsilon)$. In the first subsequence of steps of the P colony Π, only the agents B_1 and B_2 can apply their programs.

	configuration of Π								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	e	e	e	e		1	3		
2.	l_0	d	e	e			4		
3.	l_0	e	e	e	d	2	3		
4.		d	e	e	d		4		

(3) To simulate the ADD-instruction $l_1 : (ADD(r), l_2, l_3)$, there are two agents $B_{l_1^1}$ and $B_{l_1^2}$ in the P colony Π. These agents help each other to add one copy of the object a_r and the object l_2 or l_3 to the environment.

$P_{l_1^1}$	$P_{l_1^1}$	$P_{l_1^2}$
$6 : \langle e \leftrightarrow l_1 \rangle,$	$10 : \langle d \rightarrow l_3 \rangle,$	$13 : \langle e \leftrightarrow D_1 \rangle,$
$7 : \langle l_1 \rightarrow D_1 \rangle,$	$11 : \langle l_2 \leftrightarrow e \rangle,$	$14 : \langle D_1 \rightarrow a_r \rangle,$
$8 : \langle D_1 \leftrightarrow d \rangle,$	$12 : \langle l_3 \leftrightarrow e \rangle,$	$15 : \langle a_r \leftrightarrow e \rangle,$
$9 : \langle d \rightarrow l_2 \rangle,$		

The instruction $l_1 : (ADD(r), l_2, l_3)$ is simulated by the following sequence of steps. Let the content of the agent B_2 be d.

	configuration of Π						
step	B_2	$B_{l_1^1}$	$B_{l_1^2}$	Env	P_2	P_3	P_4
1.	d	e	e	$l_1 a_r^u d^v$	3	6	
2.	e	l_1	e	$a_r^u d^{v+1}$	2	7	
3.	d	D_1	e	$a_r^u d^{v+1}$	3	8	
4.	e	d	e	$D_1 a_r^u d^{v+1}$	2	**9** or 10	13
5.	d	l_2	D_1	$a_r^u d^{v+1}$	3	11	14
6.	e	e	a_r	$l_2 a_r^u d^{v+2}$	2		15
7.	d	e	e	$l_2 a_r^{u+1} d^{v+2}$	3		

(4) For each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, the below mentioned programs are introduced in the sets $P_{l_1^1}$, $P_{l_1^2}$, $P_{l_1^3}$, $P_{l_1^4}$ and in the set $P_{l_1^5}$:

$P_{l_1^1}$	$P_{l_1^1}$	$P_{l_1^2}$	$P_{l_1^5}$
$16 : \langle e \leftrightarrow l_1 \rangle$,	$19 : \langle d \rightarrow \boxed{1} \rangle$,	$21 : \langle e \leftrightarrow D_1 \rangle$,	$32 : \langle e \leftrightarrow \textcircled{1} \rangle$,
$17 : \langle l_1 \rightarrow D_1 \rangle$,	$20 : \langle \boxed{1} \leftrightarrow e \rangle$,	$22 : \langle D_1 \rightarrow \textcircled{1} \rangle$,	$33 : \langle \textcircled{1} \rightarrow 1'' \rangle$,
$18 : \langle D_1 \leftrightarrow d \rangle$,		$23 : \langle \textcircled{1} \leftrightarrow e \rangle$,	$34 : \langle 1'' \leftrightarrow e \rangle$,

$P_{l_1^3}$	$P_{l_1^3}$	$P_{l_1^5}$	$P_{l_1^5}$
$24 : \langle e \leftrightarrow \boxed{1} \rangle$,	$28 : \langle l_2 \leftrightarrow e \rangle$,	$35 : \langle e \leftrightarrow 1' \rangle$,	$39 : \langle 1'' \rightarrow e \rangle$,
$25 : \langle \boxed{1} \rightarrow 1' \rangle$,	$29 : \langle 1' \leftrightarrow 1'' \rangle$,	$36 : \langle 1' \rightarrow d \rangle$,	$40 : \langle \overline{l_3} \rightarrow l_3 \rangle$,
$26 : \langle 1' \leftrightarrow a_r \rangle$,	$30 : \langle 1'' \rightarrow \overline{l_3} \rangle$,	$37 : \langle d \leftrightarrow 1'' \rangle$,	$41 : \langle l_3 \leftrightarrow e \rangle$,
$27 : \langle a_r \rightarrow l_2 \rangle$,	$31 : \langle \overline{l_3} \leftrightarrow e \rangle$,	$38 : \langle d \leftrightarrow \overline{l_3} \rangle$,	

The instruction $l_1 : (SUB(r), l_2, l_3)$ is simulated by the following sequence of steps.

If the register r stores a value different from zero, then the computation proceeds as follows (we do not consider the number of copies of the object d in the environment):

	configuration of Π						applicable programs				
step	$B_{l_1^1}$	$B_{l_1^2}$	$B_{l_1^3}$	$B_{l_1^4}$	$B_{l_1^5}$	Env	$P_{l_1^1}$	$P_{l_1^2}$	$P_{l_1^3}$	$P_{l_1^4}$	$P_{l_1^5}$
1.	e	e	e	e	e	$l_1 a_r^u d^v$	16				
2.	l_1	e	e	e	e	$a_r^u d^v$	17				
3.	D_1	e	e	e	e	$a_r^u d^v$	18				
4.	d	e	e	e	e	$D_1 a_r^u d^{v-1}$	19	21			
5.	$\boxed{1}$	D_1	e	e	e	$a_r^u d^{v-1}$	20	22			
6.	e	$\textcircled{1}$	e	e	e	$\boxed{1} a_r^u d^{v-1}$		23	24		
7.	e	e	$\boxed{1}$	e	e	$\textcircled{1} a_r^u d^{v-1}$			25	32	
8.	e	e	$1'$	$\textcircled{1}$	e	$a_r^u d^{v-1}$			26	33	
9.	e	e	a_r	$1''$	e	$1' a_r^{u-1} d^{v-1}$			27	34	35
10.	e	e	l_2	e	$1'$	$1'' a_r^{u-1} d^{v-1}$			28		36
11.	e	e	e	e	d	$l_2 1'' a_r^{u-1} d^{v-1}$					37
12.	e	e	e	e	$1''$	$l_2 a_r^{u-1} d^v$					39
13.	e	e	e	e	e	$l_2 a_r^{u-1} d^v$					

From the 12^{th} step the agent $B_{l_2^1}$ starts to work and consumes the object l_2. We do not notice this fact in the table.

When the value in the counter r is zero:

	configuration of Π						applicable programs				
step	$B_{l_1^1}$	$B_{l_1^2}$	$B_{l_1^3}$	$B_{l_1^4}$	$B_{l_1^5}$	Env	$P_{l_1^1}$	$P_{l_1^2}$	$P_{l_1^3}$	$P_{l_1^4}$	$P_{l_1^4}$
1.	e	e	e	e	e	$l_1 d^v$	16				
2.	l_1	e	e	e	e	d^v	17				
3.	D_1	e	e	e	e	d^v	18				
4.	d	e	e	e	e	$D_1 d^{v-1}$	19	21			
5.	$\boxed{1}$	D_1	e	e	e	d^{v-1}	20	22			
6.	e	(1)	e	e	e	$\boxed{1} d^{v-1}$		23	24		
7.	e	e	$\boxed{1}$	e	e	$(1) d^{v-1}$			25	32	
8.	e	e	$1'$	(1)	e	d^{v-1}				33	
9.	e	e	$1'$	$1''$	e	d^{v-1}				34	
10.	e	e	$1'$	e	e	$1'' d^{v-1}$			29		
11.	e	e	$1''$	e	e	$1' d^{v-1}$			30		35
12.	e	e	$\overline{l_3}$	e	$1'$	d^{v-1}			31		36
13.	e	e	e	e	d	$\overline{l_3} d^{v-1}$					38
14.	e	e	e	e	$\overline{l_3}$	d^v					40
15.	e	e	e	e	l_3	d^v					41
16.	e	e	e	e	e	$l_3 d^v$					

(5) The halting instruction l_h is simulated by agent B_2 which consumes the object l_h and that stops the computation.

The P colony Π correctly simulates the computation of the register machine M. The computation of the Π starts with no object a_r, which indicates the content of the register r, placed in the environment, in the same way as the computation in the register machine M starts with zeros in all registers. Then the agents simulate the computation by simulating ADD and SUB instructions. The computation of the P colony Π stops if the symbol l_h is placed inside the corresponding agent as well as the register machine M stops by executing the halting instruction labeled l_h. Consequently, $N(M) = N(\Pi)$ and because the number of agents equals four, the proof is complete.

□

4 Modularity in the terms of P colonies

During the evolution unicellular organisms have evolved into multicellular. Some cells specialized their activities for the particular function and have to cooperate with other specialized cells to be alive. In that way the organs have evolved and living organisms have become more complex. But the cooperating organs and more complex living organisms are more sophisticated, live longer and their life is improving.

From the previous section we can observe that some agents in the P colonies are providing the same function during the computation. This inspired us to introduce the modules in the P colonies. We have defined five modules, where each of them is providing one specific function. These modules are the module for the duplication, the module for the addition, the module for the subtraction, the balance-wheel module, the control module (see Fig. 1). Definition of each module's function is given in the proof of following theorem.

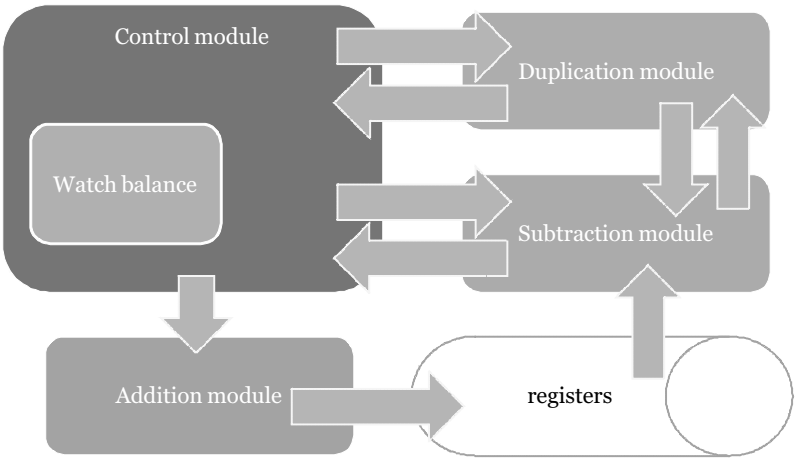


Fig. 1. Modular P colony

Theorem 3. $NPCOL_{par}(1, 8, *) = NRE$.

Proof. Let us consider a register machine M with m registers. We construct the P colony $\Pi = (A, e, f, V_E, B_1, B_2)$ simulating a computation of the register machine M with:

- $A = \{J, J', V, Q\} \cup \{l_i, l_i', l_i'', L_i, L_i', L_i'', E_i \mid l_i \in H\} \cup \{a_r \mid 1 \leq r \leq m\}$,
- $f = a_1$,
- $B_i = (O_i, P_i),\ O_i = \{e\}, i = 1, 2$

We can group the agents of the P colony into five modules. Each module needs for its work an imput and requires some objects. The result of its computation is an output:

(1) module for the duplication (uses 2 agents):

P_1 :	P_2 :
1 : $\langle e \to D_i \rangle$,	8 : $\left\langle e \leftrightarrow \boxed{D_i} \right\rangle$,
2 : $\left\langle D_i \to \boxed{D_i} \right\rangle$,	9 : $\left\langle \boxed{D_i} \to i' \right\rangle$,
3 : $\left\langle \boxed{D_i} \leftrightarrow d \right\rangle$,	10 : $\langle i' \leftrightarrow e \rangle$,
4 : $\left\langle d \to \boxed{i} \right\rangle$,	11 : $\left\langle e \leftrightarrow \boxed{i} \right\rangle$,
5 : $\left\langle \boxed{i} \leftrightarrow i' \right\rangle$,	12 : $\left\langle \boxed{i} \to \textcircled{i} \right\rangle$,
6 : $\langle i' \to i \rangle$,	13 : $\langle \textcircled{i} \leftrightarrow e \rangle$.
7 : $\langle i \leftrightarrow e \rangle$,	

Duplicating module is activated when the object D_i appears in the environment. This object carries a message "Duplicate object i.".

input: one object D_i
output: one object i after 10 steps and one object $\textcircled{i}$ after 11 steps
requirements: one object d

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	e	e	dD_i	1	–
2.	D_i	e	d	2	–
3.	$\boxed{D_i}$	e	d	3	–
4.	d	e	$\boxed{D_i}$	4	8
5.	$\boxed{i}$	$\boxed{D_i}$		–	9
6.	$\boxed{i}$	i'		–	10
7.	$\boxed{i}$	e	i'	5	–
8.	i'	e	$\boxed{i}$	6	11
9.	i	$\boxed{i}$		7	12
10.	e	$\textcircled{i}$	i	–	13
11.	e	e	$\textcircled{i}i$	–	–

Duplicating module duplicates requested object.

(2) module for the addition (uses 1 agent):

P_1 :
1 : $\langle e \leftrightarrow A_r \rangle$,
2 : $\langle A_r \to a_r \rangle$,
3 : $\langle a_r \leftrightarrow e \rangle$.

input: one object A_r
output: one object a_r after 4 steps
requirements: $\emptyset$

	configuration of Π		
	B_1	Env	P_1
1.	e	A_r	1
2.	A_r		2
3.	a_r		3
4.	e	a_r	–

Module for the addition adds one symbol into the environment.

(3) module for the subtraction (uses 3 agents):

P_1 :	P_2 :	P_3 :
1 : $\langle e \leftrightarrow S_r \rangle$,	11 : $\langle e \leftrightarrow \textcircled{B_r} \rangle$,	19 : $\langle e \leftrightarrow y' \rangle$,
2 : $\langle S_r \rightarrow D_{B_r} \rangle$,	12 : $\langle \textcircled{B_r} \rightarrow \textcircled{B'_r} \rangle$,	20 : $\langle y' \rightarrow y \rangle$,
3 : $\langle D_{B_r} \leftrightarrow d \rangle$,	13 : $\langle \textcircled{B'_r} \leftrightarrow a_r \rangle$	21 : $\langle y \leftrightarrow B'_r \rangle$,
4 : $\langle d \leftrightarrow B_r \rangle$,	14 : $\langle \textcircled{B'_r} \leftrightarrow B'_r \rangle$	22 : $\langle B'_r \rightarrow e \rangle$,

P_1 :	P_2 :	P_3 :
5 : $\langle B_r \rightarrow \underline{B}_r \rangle$,	15 : $\langle a_r \rightarrow y' \rangle$	
6 : $\langle \underline{B}_r \rightarrow \overline{B}_r \rangle$,	16 : $\langle B'_r \rightarrow n \rangle$	
7 : $\langle \overline{B}_r \rightarrow B'_r \rangle$,	17 : $\langle y' \leftrightarrow e \rangle$	
8 : $\langle B'_r \leftrightarrow d \rangle$,	18 : $\langle n \leftrightarrow e \rangle$	
9 : $\langle d \leftrightarrow \textcircled{B'_r} \rangle$,		
10 : $\langle \textcircled{B'_r} \rightarrow e \rangle$,		

input: one object B_r

output: one object y after 23 steps or one object n after 22 steps

requirements: two objects d, object a_r (if there is at least one in the environment)

uses: duplication module

	configuration of Π						
	B_1	B_2	B_3	Env	P_1	P_2	P_3
1.	e	e	e	$S_r a_r d$	1	–	–
2.	S_r	e	e	$a_r d$	2	–	–
3.	D_{B_r}	e	e	$a_r d$	3	–	–
4.	d	e	e	$a_r D_{B_r}$	–	–	–
	waiting for objects from duplication unit						
14.	d	e	e	$a_r B_r$	4	–	–
15.	B_r	e	e	$a_r \textcircled{B_r}$	5	11	–
16.	$\underline{B}_r$	$\textcircled{B_r}$	e	a_r	6	12	–
17.	$\overline{B}_r$	$\textcircled{B'_r}$	e	a_r	7	13	–
18.	B'_r	a_r	e	$\textcircled{B'_r}$	8	15	–
19.	d	y'	e	$\textcircled{B'_r} B'_r$	9	17	–
20.	$\textcircled{B'_r}$	e	e	$y' B'_r$	10	–	19
21.	e	e	y'	B'_r	–	–	20
22.	e	e	y	B'_r	–	–	21
23.	e	e	B'_r	y	–	–	22
24.	e	e	e	y	–	–	–

	configuration of Π						
	B_1	B_2	B_3	Env	P_1	P_2	P_3
1.	e	e	e	$S_r d$	1	–	–
2.	S_r	e	e	d	2	–	–
3.	D_{B_r}	e	e	d	3	–	–
4.	d	e	e	D_{B_r}	–	–	–
	waiting for objects from duplication unit						
14.	d	e	e	B_r	4	–	–
15.	B_r	e	e	(B_r)	5	11	–
16.	$\underline{B}_r$	(B_r)	e		6	12	–
17.	$\overline{B}_r$	(B'_r)	e		7	–	–
18.	B'_r	(B'_r)	e		8	–	–
19.	d	(B'_r)	e	B'_r	–	14	–
20.	d	B'_r	e	(B'_r)	9	16	–
21.	(B'_r)	n	e		10	–	–
22.	e	e	e	n	–	–	–

Module for the subtraction removes requested object from the environment.

(4) Balance-wheel module (uses 1 agent):

P_1 :

1 : $\langle e \to d \rangle$
2 : $\langle d \leftrightarrow e \rangle$
3 : $\langle d \leftrightarrow (f) \rangle$

The balance-wheel module "keeps the computation alive". It inserts the objects d into the environment until it consumes a special symbol (f) from the environment. This action makes it stop working. The object (f) gets into the environment from the duplicating module which is activated by the simulation of the halt instruction by the control module.

(5) Control module (uses 2 agents):

a) initialization:

P_1 :

1 : $\langle e \to l_0 \rangle$

First agent in this module generates label of the first instruction of the register machine.

b) adding instruction $l_1 : (ADD(r), l_2, l_3)$:

P_1 :	notes
1 : $\langle l_1 \to D_1 \rangle$,	
2 : $\langle D_1 \leftrightarrow d \rangle$,	$\longrightarrow$ Duplication module
3 : $\langle d \leftrightarrow 1 \rangle$,	$\longleftarrow$ Duplication module
4 : $\langle 1 \to B_r \rangle$,	
5 : $\langle B_r \leftrightarrow (1) \rangle$,	$\longleftarrow$ Duplication module $\longrightarrow$ Addition module
6 : $\langle (1) \to l_2 \rangle$,	
7 : $\langle (1) \to l_3 \rangle$,	

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	l_1	e	d	1	–
2.	D_1	e	d	2	–
3.	d	e	D_1	–	–
	waiting for response of duplication unit				
13.	d	e	1	3	–
14.	1	e	$d\,(1)$	4	–
15.	B_r	e	$d\,(1)$	5	–
16.	(1)	e	B_r	**6** or 7	–
17.	l_2	e	d	–	–

c) subtracting instruction $l_1 : (SUB(r), l_2, l_3)$:

P_1 :	notes	P_2 :
1 : $\langle l_1 \to D_1 \rangle$,		16 : $\langle e \leftrightarrow L_1 \rangle$,
2 : $\langle D_1 \leftrightarrow d \rangle$,	$\longrightarrow$ Duplication module	17 : $\left\langle L_1 \to \boxed{L_1} \right\rangle$,
3 : $\langle d \leftrightarrow 1 \rangle$,	$\longleftarrow$ Duplication module	18 : $\left\langle \boxed{L_1} \leftrightarrow e \right\rangle$,
4 : $\langle 1 \to S_r \rangle$,		19 : $\langle e \leftrightarrow L_1' \rangle$,
5 : $\langle S_r \leftrightarrow (1) \rangle$,	$\longleftarrow$ Duplication module $\longrightarrow$ Subtraction module	20 : $\langle\ e \leftrightarrow L_1'' \rangle$,
6 : $\langle (1) \to L_1 \rangle$,		21 : $\langle L_1' \to l_2 \rangle$,
7 : $\langle L_1 \leftrightarrow y \rangle$,	$\longleftarrow$ Subtraction module	22 : $\langle L_1'' \to l_3 \rangle$,
8 : $\langle L_1 \leftrightarrow n \rangle$,	$\longleftarrow$ Subtraction module	23 : $\langle l_2 \leftrightarrow e \rangle$,
9 : $\langle y \to L_1' \rangle$,		24 : $\langle l_3 \leftrightarrow e \rangle$,
10 : $\langle n \to L_1'' \rangle$,		
11 : $\left\langle L_1' \leftrightarrow \boxed{L_1} \right\rangle$,		
12 : $\left\langle L_1'' \leftrightarrow \boxed{L_1} \right\rangle$,		
13 : $\left\langle \boxed{L_1} \to d \right\rangle$,		
14 : $\langle d \leftrightarrow l_2 \rangle$,		
15 : $\langle d \leftrightarrow l_3 \rangle$,		

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	l_1	e	d	1	–
2.	D_1	e	d	2	–
3.	d	e	D_1	–	–
	waiting for response of duplication module				
13.	d	e	1	3	–
14.	1	e	$d\,(1)$	4	–
15.	S_r	e	$d\,(1)$	5	–
16.	(1)	e	S_r	6	–
17.	L_1	e	d	–	–
	waiting for response of subtraction module				

If subtraction module generates y

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
49.	L_1	e	y	7	–
50.	y	e	L_1	9	16
51.	L_1'	L_1		–	17
52.	L_1'	$\boxed{L_1}$		–	18
53.	L_1'	e	$\boxed{L_1}$	11	–
54.	$\boxed{L_1}$	e	L_1'	13	19
55.	d	L_1'		–	21
56.	d	l_2		–	23
57.	d	e	l_2	14	–
58.	l_2	e	d	–	–

If subtraction module generates n

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
48.	L_1	e	n	8	–
49.	n	e	L_1	10	16
50.	L_1''	L_1		–	17
51.	L_1''	$\boxed{L_1}$		–	18
52.	L_1''	e	$\boxed{L_1}$	12	–
53.	$\boxed{L_1}$	e	L_1''	13	20
54.	d	L_1''		–	22
55.	d	l_3		–	24
56.	d	e	l_3	15	–
57.	l_3	e	d	–	–

d) halting instruction l_h:

P_1 :
1 : $\langle l_f \rightarrow D_f \rangle$
2 : $\langle D_f \leftrightarrow d \rangle$ $\longrightarrow$ Duplication module
3 : $\langle d \leftrightarrow f \rangle$ $\longleftarrow$ Duplication module

Control module controls all the computation. It sends necessary objects into the environment for the work of the other modules.

The P colony Π correctly simulates any computation of the register machine M. □

5 Conclusion

We have proved that the P colonies with capacity $k = 2$ and without checking programs with height at most 2 are computationally complete. In Section 3 we have shown that the P colonies with capacity $k = 1$ and with checking/evolution programs and 4 agents are computationally complete.

We have also verified that partially blind register machines can be simulated by P colonies with capacity $k = 1$ without checking programs with two agents. The generative power of $NPCOL_{par}K(1, n, *)$ for $n = 2, 3$ remains open.

In Section 4 we have studied P colonies with capacity $k = 2$ without checking programs. Two agents guarantee the computational completeness in this case.

Remark 1. This work has been supported by the Grant Agency of the Czech Republic grants No. 201/06/0567 and SGS/5/2010.

References

1. Ciencialová, L., Cienciala, L.: *Variations on the theme: P Colonies*, Proceedings of the 1^{st} International workshop WFM'06 (Kolář, D., Meduna, A., eds.), Ostrava, 2006, pp. 27–34.
2. Ciencialová, L. Cienciala, L., Kelemenová, A.: *On the number of agents in P colonies*, In G. Eleftherakis, P. Kefalas, and G. Paun (eds.), *Proceedings of the 8th Workshop on Membrane Computing (WMC'07)*, June 25-28, Thessaloniki, Greece, 2007, pp. 227–242.
3. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, G.: *Cells in environment: P colonies*, Multiple-valued Logic and Soft Computing, 12, 3-4, 2006, pp. 201–215.
4. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: *P Colonies with a bounded number of cells and programs.* Pre-Proceedings of the 7^{th} Workshop on Membrane Computing (H. J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), Leiden, The Netherlands, 2006, pp. 311–322.
5. Freund, R., Oswald, M.: *P colonies working in the maximally parallel and in the sequential mode.* Pre-Proceedings of the 1^{st} International Workshop on Theory and Application of P Systems (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, pp. 49–56.
6. Greibach, S. A.: *Remarks on blind and partially blind one-way multicounter machines.* Theoretical Computer Science, 7(1), 1978, pp. 311–324.
7. Kelemen, J., Kelemenová, A.: *On P colonies, a biochemically inspired model of computation.* Proc. of the 6^{th} International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, pp. 40–56.
8. Kelemen, J., Kelemenová, A., Păun, Gh.: *Preview of P colonies: A biochemically inspired computing model.* Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE IX (M. Bedau at al., eds.) Boston, Mass., 2004, pp. 82–86.
9. Minsky, M. L.: *Computation: Finite and Infinite Machines.* Prentice Hall, Engle-wood Cliffs, NJ, 1967.
10. Păun, Gh.: *Computing with membranes.* Journal of Computer and System Sciences 61, 2000, pp. 108–143.
11. Păun, Gh.: *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
12. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.
13. P systems web page: http://psystems.disco.unimib.it

A New P System to Model the Subalpine and Alpine Plant Communities

Maria Angels Colomer[1], Cristian Fondevilla[1], and Luis Valencia-Cabrera[2]

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

Summary. In this work we present a P system based model of the ecosystem dynamics of plant communities. It is applied to four National Hunting Reservoirs in Catalan Pyrenees (Spain). In previous works several natural high- mountain- ecosystems and population dynamics were modeled, but in those works grass was considered unlimited and changes in plant communities were not taken into account. In our new model we take advantage of the modularity of P systems, adding the plant communities to an existing model on scavengers dynamics [6]. We introduce the plant community production and two possible changes or evolutions in communities: (1) due to less grazing pressure, and (2) due to recovering pastures with human management as for example fire or clearing.

1 Introduction

Everyday the knowledge on the functioning of biological processes and involved variables gain more importance. It is interesting to model this acquired knowledge through parallel independent work, defining a model as close to reality as possible, allowing to simulate behavioural processes in different possible scenarios.

Natural processes have a great complexity, because each biological process involves a large number of variables and their interactions, so that modelling ecosystems is not a simple task [5]. Classical models such as differential equations present several limitations. Most of all they cannot consider simultaneously multiple species and their interactions. The most frequently used models are viability models or multi-agent models, that do not allow the study of different species with their interactions. In the work at hand, we make use of bio-inspired models, called P systems, which could serve as an alternative to the multi-agent models, having in mind that in the case of the P systems the evolution rules cannot be expressed mathematically.

The use of P systems to model these natural processes has important advantages, such as the modularity (P systems can be composed of modules, which make it easier to modify and improve existing models). Another important characteristic of P systems is their ability to work parallel. Also, because strictly mathematical expressions are not required, there is no limitation to the number of variables. These characteristics make P systems very helpful for modelling complex ecosystems [6]. Their problem is, just like for every complex model, the necessity to hold an extensive set of experimental data to be successful. The more complex is the process to study, the more knowledge is needed. For this reason it is required to work with interdisciplinary teams of experts.

P systems are very useful to model natural ecosystems. Their properties make them very attractive for modelling complex ecosystems. Recently some works that model ecosystems using P Systems have been published [3, 4, 6]. In these works models focusing on animal population dynamics were presented. They model natural processes as feeding, grazing, reproduction and mortality. The food source is grass for ungulate species and meat or bones for avian scavengers.

In the present work, the amount of grass available for grazing is not a fixed value, as it was in previous ones. It depends of a large number of factors and furthermore of the existence of plant communities which evolve due to their management. The aim is to define new modules, which are incorporated into the model presented for the avian scavengers [6]. These modules operate parallel with the modules of the existent scavengers model.

We have developed a model in which different plant communities produce an amount of grass, in function on climate variables. In addition the plant communities can change and evolve over time due to changes in biotic or abiotic processes. Generally the abiotic processes occur in higher altitudes, and are not sufficiently known to be understood and to be modelled. The biotic processes play a more important role, causing changes in plant communities, in particular due to the grazing management.

2 Alpine and Subalpine Plant Communities

In the plant-geographic sense alpine life zone exclusively encompasses vegetation above the natural high altitude treeline [10]. In our work we also consider the subalpine life zone above the actual treeline. Both zones can be defined as man made pastures for summer grazing above the actual treeline.

The Phytosociology is the science that analyses and characterises the different levels of plant associations called formally "Plant Communities". These communities are divided into classes, these consist of orders, these of alliances and the latter of associations. The alliance level was chosen as the focus of this work, because of its better availability of information.

In high altitude zones the vegetation activity does not occur throughout the whole year, but is concentrated in the months in which the conditions are

favourable. Thus, the growing season, is the part of the year when the vegetation is active and productive [10].

2.1 Communities production

Production is the amount of new biomass accumulation over a longer period of time.[10].

There are many methods to quantify the aerial net primary production (ANP) in grasslands; Singh et al. [15] provide in their work a list of exhaustive methods that have applied in the following several authors [8, 11]. In the present work we have chosen the method defined as peak standing crop of current live material plus recent dead, because it is the most commonly used and allows comparisons. This method includes the estimate of ANP, the peak community biomass (weight of the live vegetation) plus the weight of current season's growth which has reached the senescence before the date of peak live biomass [15].

2.2 Grassland dynamics. Plant Communities changes

This concept refers to changes over time of grassland structures and the replacement of some plant communities by others in relation to the variation of environmental factors, including animal grazing and human management [9]. Thus, grasslands are not static, but they evolve and transform over time due to abiotic and biotic factors. In our current work we analyse two changes in plant alliances: (1) the evolution due to less grazing pressure and, the inverse case, (2) the evolution due to recovering pastures with human management. However there are many other possible changes to treat. The observation and the study of these changes, some of which are still quite unknown [9], give the model wide possibilities for future research.

There are two different types of changes in plant communities: biotic and abiotic processes. The two situations taken into consideration in our work are located within the second group, inside the landscape changes due to anthropic management below 2000 meters.

Some plant communities, represented here by plant alliances, are not stable, but were created thanks to the use of these surfaces by cattle. Thus, very important ecological communities with a high degree of biodiversity have emerged. They increase the visual quality of landscape, without forgetting its significance for animal grazing, both domestic and wild. Therefore its protection, conservation and study are of importance.

Changes due to less grazing pressure are the most important ones that occur in our days in alpine and subalpine ecosystems, upper 1500 meters. Nowadays exist less domestic animals that exploit these grasslands, or in other words the currently carrying capacity is higher than the real number of animals.

In this case the grassland evolves first to a bush land and then to a coniferous forest (i.e. Class. *Vaccinio-Piceetea* Br.-Bl. in Br.-Bl., Sissingh & Vlieger

1939). In this first version of our work only the transformation from grassland (Al. *Bromion erecti* Koch 1926) to scrubland (Al. *Juniperion nanae* Br.-Bl. et al. 1939) is introduced into the model.

The alliance *Bromion erecti* can be defined as Pyrenean meso-xerophytic grassland; and *Juniperion nanae* as Alpine-Pyrenean dwarfed shrub and heaths of windswept places without long snow cover and sunny expositions [14].

The evolution of the vegetation due to human management to recover grasslands (e.g. fire, clearing) is the opposite of the previous case occurred due a low livestock pressure. In this processes the shrub *Juniperion nanae* evolve to the grassland *Bromion erecti*. The most likely cause is the controlled burning of these areas for grazing.

3 Materials and methods

The National Hunting Reserves of Catalonia (Spain) (NHR) are geographically defined territories, with special characteristics, declared to promote, preserve and protect native fauna species. They are located in mountainous areas with high ecological and landscape quality. In these areas exists a very important wildlife fauna, including some species of great significance for hunting. In this work we have chosen the four National hunting reserves placed in the Catalan Pyrenees: Alt Pallars-Aran (106.661 ha), Cerdanya-Alt Urgell (19.003 ha), Cadí (48.449 ha) and Freser-Setcases (20.200 ha) (data provided by [12], see figure 1). Over the centuries, these alpine and subalpine zones have been exploited by man to feed their herds, sheep, cows and mares and themselves, and their management has contributed to the landscape transformation. The Pyrenean orography presents altitudinal zones between 1000 and 3000 m but we use only the higher zones between 1500 and 3000 m, occupied mostly by grass.

In the Pyrenean region, the level of annual rainfall ranges from 800 to 1200 mm and the maximum average temperatures do not exceed 25 ℃ in summer and do not fall under 5 ℃ in winter. To characterise it, we provide a series of temperatures and precipitation obtained from different weather stations located in the study area provided by the Meteorological Service of Catalonia.

To model the high mountain ecosystem as correctly as possible, we have separated our range (from 1500 to 3000 meters) into three different zones in function of their height due to their different climatic and orographic characteristics, that define which plant communities are established in each range. As a result we have defined a low altitude zone (from 1500 to 2000 m), an intermediate zone (from 2000 to 2500 m) and an upper zone (more than 2500 m). The first and the second range zone can be considered as low and high subalpine zones, while the third is alpine altitudinal range. This distinction is related to different plant communities available in each range, their primary production and the different length of their seasonal growing periods, as the growing season in low ranges is longer than in the high one.

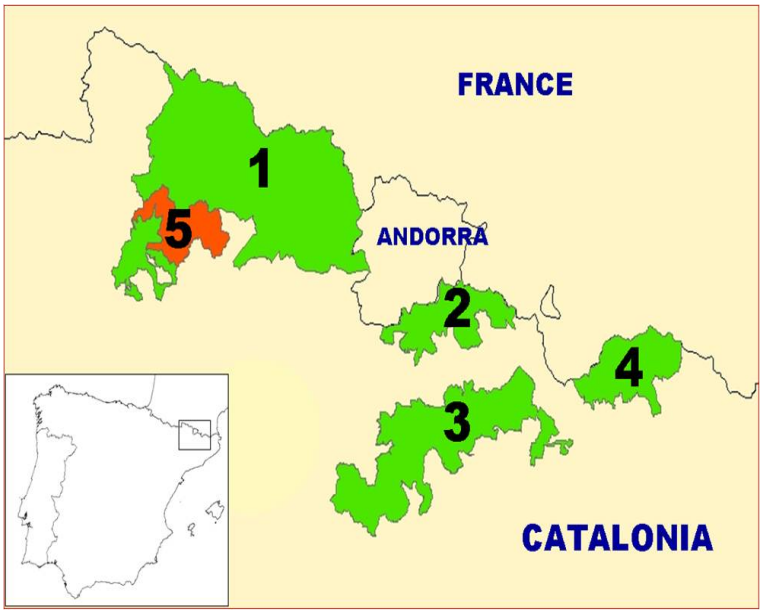


Fig. 1. Study area in the Catalan Pyrenees. Area 1: National Hunting Reservoir in l'Alt Pallars-Aran. Area 2: RNC Cerdanya-Alt Urgell. Area 3: RNC Cadí. Area 4 RNC: Freser-Setcases. Area 5: National Park, not include in the study.

Based on vegetation maps available [7], we defined the areas corresponding to the four National Hunting Reserves (RNC) and areas with an altitude below 1.500 meters were eliminated. As a result, we get the entire area bounded by the RNC situated above 1.500 meters, corresponding to the subalpine and alpine biogeographic regions.

In the present work we have identified a total of 26 different plant alliances (table 1). Each plant alliance is associated to an average production and a standard deviation [2, 1, 8, 11]. Each Natural Hunting Reservoir has a determinate surface at each altitude level, obtained from vegetation maps at scale 1:50000 [7].

The plant community production depends on:

- Type of plant community; in this work we focus on the alliance level.
- Weather, that determine the community production and the length of the growing season.
- Altitude, closely related to the previous concept. At higher altitudes, we have lower temperatures and shorter growing season.

An important variable is the length of the growing season. It is determined by climate and elevation. The beginning of the growing season is defined by means of the thermal integral, being the starting day of this period, for each altitudinal

range, in which the sum of positive daily temperatures from the beginning of the year exceed 300 ℃. The end of the period is set from the consulted references [11].

Table 1. The 26 alpine and subalpine plant communities (at alliance level) defined in our model, their primary production ($g \cdot m^{-2} \cdot day^{-1}$) and their standard deviation [2, 1, 11] and their altitude range [9].* These alliances are assimilated to other similar alliances. ** Shrub community, not available for grazing.

Plant community	**Production**	**Standard deviation**	Altitude
Al.*Aphyllantion*	*	*	Low
Al.*Arabidion caeruleae*	*	*	Medium, High
Al.*Arrhenatherion-Bromion*	4,322	0,395	Low
Al.*Arrhenatherion 1*	7,253	0,662	Low
Al.*Arrhenatherion 2*	4,403	0,402	Low
Al.*Arrhenatherion 3*	3,529	0,322	Low
Al.*Bromion erecti*	4,026	0,368	Low, Medium
Al.*Bromion-Nardion*	4,026	0,410	Low, Medium
Al.*Caricion davallinae*	*	*	Low, Medium
Al.*Caricion nigrae*	2,966	0,271	Low, Medium
Al.*Cynosurion cristati*	2,292	0,209	Low
Al.*Elyonion myosuroidis*	1,750	0,160	Medium, High
Al.*Festuca panniculata*	3,895	0,356	Low, Medium
Al.*Festucion airoidis*	*	*	Medium, High
Al.*Festucion eskiae*	4,538	0,414	Low, Medium, High
Al.*Festucion scopariae*	0,967	0,088	Low, Medium, High
Al.*Juniperion nannae*	**	**	Low
Al.*Nardion strictae*	3,091	0,282	Low, Medium
Al.*Ononidion striatae*	4,028	0,368	Low, Medium
Al.*Polygonion avicularis*	3,266	0,298	Low, Medium
Al.*Primulion intricatae*	2,472	0,226	Low, Medium
Al.*Rumicion pseudoalpini*	3,26	0,298	Low, Medium
Al.*Salicion herbaceae*	1,292	0,118	Medium, High
Al.*Saponarion cespitosae*	*	*	Low, Medium, High
Cl.*Thlaspietea rotundifolii*	0,041	0,004	Low, Medium, High
Al.*Triseto-Polygonion*	2,795	0,255	Low, Medium

4 A P System Based modeling framework

In this section, we define a P system based framework where additional features, such as probabilistic functions and three electrical charges that better describe specific properties, are used.

A *skeleton of an extended P system* with active membranes of degree $q \leq 1$, $\Pi = (\Gamma, \mu, R)$, can be viewed as a set of (polarised) membranes hierarchised by a structure of membranes μ (a rooted tree) labeled by $0, 1, \ldots, q-1$. All

membranes in μ are supposed to be (initially) neutral and they have associated with them R, a finite set of evolution rules of the form $u[v]_i^{\alpha} \rightarrow u'[v']_i^{\beta}$ that can modify their polarisation but not their label. Γ is an alphabet that represents the objects (i.e., animals, plant alliances, etc., see Fig. 3).

A probabilistic functional extended P system with active membranes of degree $q \leq 1$ taking T time units, $\Pi = (\Gamma, \mu, R, T, \{f_r : r \in R\}, M_0, \ldots, M_{q-1})$, can be viewed as a skeleton (Π, μ, R) with the membranes hierarchized by the structure μ labeled by $0, 1, \ldots, q-1$. T is a natural number that represents the simulation time of the system. For each rule $r \in R$ and a, $1 \leq a \leq T$, $f_r(a)$ is a whole number between 0 and 1, which represents a probabilistic constant associated with rule r at moment a. In a generic way, we denote $r : u[v]_i^{\alpha} \stackrel{f_r(a)}{\rightarrow} u'[v']_i^{\alpha'}$.

The tuple of multisets of objects present at any moment in the q regions of the system constitutes the configuration of the system at that moment. The tuple $(M_0, , \ldots, M_{q-1})$ is the initial configuration of Π.

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

- A rule $r : u[v]_i^{\alpha} \stackrel{f_r(a)}{\rightarrow} u'[v']_i^{\alpha'}$ is applicable to a membrane labeled by i, and with α as electrical charge if multiset u is contained in the membrane immediately outside of membrane i, it is to say membrane father of membrane i, and multiset v is contained in the membrane labeled by i having α as electrical charge. When that rule is applied, multiset u (respectively v) in the father of membrane i (respectively in membrane i) is removed from that membrane, and multiset u' (respectively v') is produced in that membrane, changing its electrical charge to α'.
- $M(\Gamma)$ is the set formed by the multisets of Γ. If $u, v \in M(\Gamma)$, $i \in \{0, \ldots, q-1\}$, $\alpha \in \{0, +, -\}$ and $r_1, \ldots r_z$ are the rules applicable whose left-hand side is $u[v]_i^{\alpha}$ at given moment a, then it should be verified that $f_{r1}(a) + \cdots + f_{rz}(a) = 1$, and the rules will be applied according to the corresponding probabilities $fr1(a), \ldots, f_{rz}(a)$.

A multienvironment functional probabilistic P system with active membranes of degree (q,m) with $q \geq 1$, $m \geq 1$, taking T time units $T \geq 1$.
$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,j} : r \in R_{\Pi}, 1 \leq j \leq m\}, \{M_{i,j} : 0 \leq i \leq q-1, 1 \leq j \leq m\})$

can be viewed as a set of m environments $e-1, \ldots, e_m$ linked by the arcs from the directed graph G. Each environment e_j contains a probabilistic functional extended P system with active membranes of degree q, $\Pi = (\Gamma, \mu, R, T, \{f_{r,j} \in R_{\Pi}, 1 \leq j \leq m\}, M_{i,j} : 0 \leq i \leq q-1, 1 \leq j \leq m)$ each of them with the same skeleton, $\Pi = (\Gamma, \mu, R)$, and such that $M_{0j}, \ldots, M_{q-1j}$ describes their initial multisets. Σ is an alphabet that represents the objects of Γ that can be present in the different environments.

The communication rule between environments in R_E are of the form $r_e : (x)_{ej} \stackrel{p_{x,j,k}}{\rightarrow} (y)_{ek}$, and for each $x \in \Sigma, 1 \leq j \leq m, 1 \leq a \leq T$, it verifies $\sum P_{x,j,k}(a) = 1$. When a rule of this type is applied the object x moves from

environment e_j to environment e_k converted into y, according to the probability $p_{j,k}$.

We assume that a global clock exists, marking the time for the whole system (for its compartments), that is, all membranes and the application of all rules are synchronized. In the P systems, a configuration consists of multisets of objects present in the m environments and at each of the regions of the P systems located in the environment.

The P system can pass from one configuration to another by using the rules from $R = R_E \cup \bigcup_{j=1}^{m}$ as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, and all applicable rules are simultaneously applied and all occurrences of the left-hand side of the rules are consumed, as usual.

5 Model

In order to model this ecosystem we use a *multienvironment functional probabilistic P system with active membranes* of degree (5,5) (five membranes and five environments), taking T time units (simulation years). We model the population dynamics of 13 animal species ($N = 13$) with 26 different plant communities ($NA = 26$) in 5 different environments ($E = 5$).

$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,j} : r \in R_\Pi, 1 \leq j \leq 5\}, \{M_{i,j} : 0 \leq i \leq 4, 1 \leq j \leq 5\})$

We have 5 environments, 4 of them associated to a each National Hunting Reservoir and a fifth environment, in which occur the processes under a lack of resources.

Where:

1. The graph of the system is $G = (\phi)$ because, in this case, there are no animal movements between environments.
2. The membrane structure is

   $\mu = [[\quad]_1[\quad]_2[\quad]_3[\quad]_4]_0$

   The first three membranes are associated with the altitude: low, medium and high altitudinal ranges.

   The initial configuration is:

   $$\begin{array}{ll} Environment : & \{T, R\} \\ Membranes : & M_0 = \{P_0, d_{m,i} : 1 \leq m \leq 3, 1 \leq i \leq N\}, \\ & M_i = \{X_{ij}, A_v, U, \rho_0, \beta : 1 \leq i \leq N, 0 \leq j \leq g_{i,6}, 1 \leq v \leq NA\} \end{array}$$

3. The working alphabet of the P system is

$$\begin{aligned}\Gamma = \{&X_{ij}, Y_{ij}, Z_{ij}, Z'_{mij}, Z''_{kmij}, WN_{mij}, V_{kmij}, V'_{mij}, W'_{kmij} : \\ &1 \le i \le N, 0 \le j \le g_{i,6}, 1 \le k \le E, 1 \le m \le 3\} \cup \\ &\{d_m i, a_i, e_i, e'_{mi}, a'_{mi}, a''_{kmi} : 1 \le i \le N, 1 \le m \le 3, 1 \le k \le E\} \cup \\ &\{A_v, G_v, G'_{mv}, G''_{kmv} : 1 \le v \le NA, 1 \le m \le 3, 1 \le k \le E\} \cup \{U, \beta, \gamma, \alpha\} \cup \\ &\{D_i, H_i, C_i : 1 \le i \le N\} \cup \\ &\{B, B'_m, B'_{km}, M, M'_m, M'_{km} : 1 \le m \le 3, 1 \le k \le E\} \cup \\ &\{P_i, \rho_i : 0 \le i \le 15\}\end{aligned}$$

Objects X_{ij}, Y_{ij}, Z_{ij}, Z'_{mij}, Z''_{kmij}, WN_{mij}, V_{kmij}, V'_{mij}, W'_{kmij} represent the same animal but in different states. Objects B, B'_i, B''_{km} and H_i, represent bones, and M, M'_i, M''_{km} and C_i represent meat left by specie i. By the objects d_i, a_i, a'_{mi}, a'_{kmi}, e_i and e'_{mi} is controlled the maximum number of animals per species in the ecosystem. D_i is an object used to count the existing animals of specie i. If a species overcomes the maximum density values, it will be regulated. In all these objects index i is associated with the type of animal, index j is associated with the age, and $g_{i,6}$ is the average life expectancy, k is the environment and m the altitudinal range. A_v is a surface unit of the alliance v and G_v, G''_{vm}, G''_{kmv} is the amount of grass produced per hectare by alliance (A_v) in each altitudinal range m and environment k. U is an object used to control the carrying capacity, and objects β, α and γ are used to determine the grazing pressure level in the environment. At the end, objects P_i and ρ_i are counters that allow the synchronization of the P system. Necessary parameters introduced into the model to model the plant communities' dynamics, is given in table 2. The parameters related to animal dynamics can be consulted in Colomer et al. [6].

4. The environment alphabet is

$$\begin{aligned}\Sigma = \{&R, T\} \cup \{T'_{k,j}, R'_{k,s}, N_{j,s} : 1 \le k \le E, 1 \le j \le 100, 1 \le s \le 100\} \cup \\ &\{G'_{m,v}, G''_{k,m,v} : 1 \le k \le E, 1 \le m \le 3, 1 \le v \le NA\} \cup \\ &\{B'_m, B''_{k,m}, M'_m, M''_{k,m} : 1 \le k \le E, 1 \le m \le 3\} \cup \\ &\{a'_{m,i}, a''_{k,m,i}, Z'_{m,i,j}, Z''_{k,m,i,j} : 1 \le k \le E, 1 \le m \le 3, 1 \le i \le N, 0 \le j \le g_{i,6}\}.\end{aligned}$$

T and R are objects that include the climatic variability, T for the length of the growing season and R for the production of plant communities. The object N carries both information. All other objects belong to Γ and have been discussed in the respective sections.

5. The set R_E and R_Π is presented in the Appendix.

The model is structured in 8 modules, the scheme appears in the figure 2 and the details of rules in the Appendix. In the following the different modules are described.

5.1 Animal modules

The modules referring to the population dynamics (reproduction, mortality, feeding and density regulation and change in the environment module) were explained in detail in previous works [5, 6], therefore we only give a brief summary of all these modules in this work.

Table 2. Parameters that affect animals and plant communities dynamics (v animal specie, i plant community (alliance), m altitudinal range, k environment (NHR)).

Climatic variables	**Parameter**
Random numbers	$1 \leq NZ \leq 100$
Animals	**Parameter**
Equivalent weight	ew_v, $1 \leq v \leq 13$
Plant communities	**Parameter**
Amount of grass produced dairy (net primary production NPP)	μR_i, $1 \leq i \leq 26$
Standard deviation of plant net primary production	σR_i, $1 \leq i \leq 26$
Carrying capacity	Ca_k, $1 \leq k \leq 4$
Mean of the growing season length	μT_i, $1 \leq i \leq 26$
Standard deviation of growing season length	σT_i, $1 \leq i \leq 26$
Surface at low, medium and high altitude	as_i, $1 \leq as \leq 3$
Abandoned land evolution	ta
Fire evolution	fe
Fire evolution probabilities	fp
Ecosystem (plant community, altitude range, environment)	$\delta_{i,m,k}$, $1 \leq i \leq 26, 1 \leq m \leq 3, 1 \leq k \leq 4$

Reproduction module

At the beginning an object of type X is associated with each animal. When rules from the reproduction module are applied to objects of type X, they evolve into objects of type Y. Objects associated with females that reproduce create new objects Y at age 0 ($Y_{i,0}$) and evolve to the object Z with the same index. The rules applied in this module are of the type: $[X_{i,j}]_h^\alpha \xrightarrow{fr} [X_{i,j}, X_{i,0}]_h^\alpha$.

Mortality module

Two different mortality causes are considered, natural mortality and hunting mortality. When the domestic animals reach their life expectancy, they do not die, but they are removed from the ecosystem. when the animals die in the ecosystem and their bodies are not removed, they leave biomass, meat (C, M) and bones, (B, H). Mortality rules are of the form:

- When they leave biomass:

  $[Y_{i,j}]_h^\alpha \xrightarrow{fr} [H_i, C_i, B, M]_h^\alpha$.

- When they are removed from the ecosystem:

  $[Y_{i,g_{i,6}}]_h^\alpha \xrightarrow{fr} [\#]_h^{\alpha-}$.

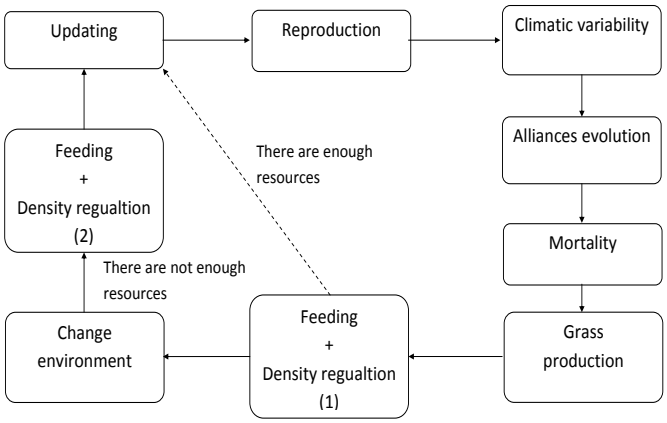


Fig. 2. Scheme model of the plant community and animal dynamics model.

Feeding and density regulation module

Whether or not the maximum carrying capacity of the ecosystem for each species has been reached, is determined by using objects a previously generated. Each altitudinal range of each environment and animal specie has associated its own carrying capacity.

Furthermore, objects Y evolve to objects Z to begin the feeding process. In the second step of this module, objects Z evolve to objects W if there is enough physical space and food.

If there are not enough resources for animals, objects Z leave the environment.

5.2 Plant community modules

The following modules have been added to improve and complete the modelling of the ecosystem presented by Colomer et al. [6]. It consists of three separate modules, the first one incorporates the production of grass that includes the grazing process, and the second which is dedicated to the changes and developments that occur in plant communities, in our ecosystem at alliance level. Finally we included also a preliminary module to introduce the climatic variability into the model.

Climate variability module

With the aim to introduce the climatic variability in the model, the objects T and R are used, whereat T includes the variability of the duration of the growing season, and R the production of communities. Previously, we created a set of 100 random numbers following a normal distribution with mean 0 and standard deviation 1. In the environment labeled as 1, T object evolves to four objects of type $T'_{k,j}$, and these are sent to their respective environments (2,3,4). The rest of T objects, placed in the environments 2, 3 and 4 disappear at the same moment. The same occurs with the object R. $T'_{k,j}$ and $R'_{k,s}$ evolve in each environment into a new object $N_{m,j,s}$, which contains the information of both objects T and R and enters into the membrane m ($1 \leq j \leq 100, 1 \leq s \leq 100, 1 \leq k \leq E, 1 \leq m \leq 3$).

Plant communities production and grazing modules

The following rule produces an amount of grass, according to the information included into the object $N_{j,s,m}$. Thus, when the object $N_{j,s,m}$ and the object A_i (a surface unit of the alliance i) come together, they produce an amount of grass G_i available for herbivores.

Once food is produced, it serves as food for different species of ungulates present in the ecosystem. In this process it is possible to obtain two different scenarios: (1) the animal has enough available food and feeds, and (2) the animal cannot find food and goes to another high range. In the first one the animal (object Z) eats, and transforms itself into object Wn. In the second case the environmental module is applied as is explained in the following section.

Change environment module

When the animals cannot find enough resources, they leave their zone (membrane) with the following set of objects (G', a', B', M') and go to a membrane 4 of an environment 5. From there, objects that represent the animals can find resources and evolve into an object W' or evolve directly into an object V. These objects return to their environment and membrane and objects W' evolve into object WN, objects V evolve into V' and disappear later on creating objects of type B, M, C, H.

Plant communities evolution

Two different processes are introduced into the model encompassed into the evolution of the plant communities due to management: (1) less grazing pressure, and (2) recovering pastures with fire.

In the first case the object U controls the minimal carrying capacity required to maintain the type of grassland or plant community. If there is no abandonment, an object α is created, in the contrary case the β object evolves to gamma and

changes the charge of the membrane to positive. When it occurs the alliance A_i can evolve into an alliance A_j with a stated probability.

When the surface of the alliance A_j scrubland exceeds a certain value, this plant community can evolve to grassland A_i with a stated probability due to human management (in this case fire).

Updating module

This module has the aim to restore the initial configuration in order to start a new simulation.

6 Final considerations

In this work we have presented a model to simulate the grassland dynamics, which allow to simulate behaviour in different scenarios. The next step is to define the simulator with MeCoSim [13] to validate the results. Afterwards we will improve the model by introducing new possible plant communities' evolutions.

Acknowledgement

We thank Ricardo García-González and Daniel Gómez for data provided and Federic Fillat for his help and contributed ideas throughout the work.

Appendix

Rules of the model.

- Counters
- In order to synchronize the model are needed objects that act as counters.

$$r_0 \equiv [\rho_i]_m^0 \to [\rho_{i+1}]_m^0, \quad \begin{cases} 0 \leq i \leq 11, \\ i <> 5, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_1 \equiv [\rho_i]_m^0 \to [\rho_{i+1}]_4^0, \quad \begin{cases} 0 \leq i \leq 12, \\ i <> 10. \end{cases}$$

$$r_2 \equiv [P_i]_0^0 \to [P_{i+1}]_0^0, \quad \{ 0 \leq i \leq 14.$$

- Density control.

$$r_3 \equiv [d_{m,i}]_0^0 \rightarrow d_{m,i}[\quad]_0^0, \quad \begin{cases} 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_4 \equiv d_{m,i}[\quad]_0^0 \rightarrow [d'_{m,i}, a_i'^{0.9d_{i1mk}}, e_{m.i}'^{0.2d_{i1mk}}]_0^0, \quad \begin{cases} 1 \leq j \leq 2, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3, \\ 1 \leq k \leq E. \end{cases}$$

$$r_5 \equiv d'_{m,i}[\quad]_m^- \rightarrow [d'_{m,i}]_m^0, \quad \begin{cases} 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_6 \equiv [d'_{m,i}]_m^- \rightarrow d_{m,i}[\quad]_m^0, \quad \begin{cases} 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

Reproduction module

- Males that do not reproduce.

$$r_7 \equiv [X_{i,j} \overset{(1-k_{i,1})}{\rightarrow} Y_{i,j}]_m^0, \quad \begin{cases} 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

- Females at fertile age that reproduce.

$$r_8 \equiv [X_{i,j} \overset{k_{i,1} \cdot k_{i,2}}{\rightarrow} Y_{i,j}, Y_{i,0}^{k_{i,3}}]_m^0, \quad \begin{cases} g_{i,3} \leq j < g_{i,4}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

- Females at fertile age that do not reproduce.

$$r_9 \equiv [X_{i,j} \overset{k_{i,1} \cdot k_{i,2}}{\rightarrow} Y_{i,j}]_m^0, \quad \begin{cases} g_{i,3} \leq j < g_{i,4}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

- Adult non-fertile males and females.

$$r_{10} \equiv [X_{i,j} \rightarrow Y_{i,j}]_m^0, \quad \begin{cases} g_{i,4} \leq j \leq g_{i,5}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

- Non-fertile young animals

$$r_{11} \equiv [X_{i,j} \rightarrow Y_{i,j}]_m^0, \quad \begin{cases} 0 \leq j < g_{i,3}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{12} \equiv [X_{i,j} \rightarrow Y_{i,j}]_m^0, \quad \begin{cases} 0 \leq j < g_{i,3}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

Climatic variability module

- Generate randomise climatic variables

$$r_{e1} \equiv (T \overset{\frac{1}{100}}{\to} T'_{1,j}, T'_{2,j}, T'_{3,j}, T'_{4,j})_{e1}, \quad \{ 1 \le j \le 100,$$

$$r_{e2} \equiv (R \overset{\frac{1}{100}}{\to} R'_{1,s}, R'_{2,s}, R'_{3,s}, R'_{4,s})_{e1}, \quad \{ 1 \le s \le 100,$$

$$r_{e3} \equiv (T \to \#)_k, \quad \{ 2 \le k \le E,$$

$$r_{e4} \equiv (R \to \#)_k, \quad \{ 2 \le k \le E,$$

Plant communities evolution module

Less grazing pressure

- If do not exist abandonment an object α is created.

$$r_{13} \equiv [\beta, U^{(Ca_{m,k} \cdot \delta_{m,k})}]^0_m \overset{a}{\to} [\alpha]^0_m, \quad \begin{cases} 1 \le i \le 13, \\ 1 \le m \le 3, \\ 1 \le k \le E. \end{cases}$$

- If the abandonment exist the object β evolve to γ.

$$r_{14} \equiv [\beta]^-_m \to [\gamma]^0_m, \quad \{ 1 \le m \le 3.$$

- The alliance evolve if during a time period has been abandoned.

$$r_{15} \equiv [\alpha, \gamma]^0_m [\#]^0_m, \quad \{ 1 \le m \le 3.$$

- If there are the conditions, the membrane charge change.

$$r_{16} \equiv [\rho_{12}, \gamma \cdot Ca]^0_m \to [\rho_{12}]^+_m, \quad \{ 1 \le m \le 3.$$

- The alliances evolve.

$$r_{17} \equiv [A_i]^+_m \overset{p_{i,j}}{\to} [A_j]^0_m, \quad \begin{cases} 1 \le m \le 3, \\ 1 \le i \le NA, \\ 1 \le j \le NA. \end{cases}$$

Human management

- When the surface of alliance i exceeds a certain value it evolves to the alliance j with a given probability.

$$r_{18} \equiv [A^{fe}_i]^0_m \overset{fp}{\to} [A^{fe}_j]^0_m, \quad \begin{cases} 1 \le i \le 26, \\ 1 \le j \le 26, \\ 1 \le m \le 3. \end{cases}$$

$$r_{19} \equiv [A^{fe}_i]^0_m \overset{1-fp}{\to} [A^{fe}_i]^0_m, \quad \begin{cases} 1 \le i \le 26, \\ 1 \le m \le 3. \end{cases}$$

Mortality module

- Young animals that survive.

$$r_{20} \equiv [Y_{i,j}]^0_l \overset{1-m_{i,1}-m_{i,3}}{\to} [Z_{i,j}, D_i]^0_l, \quad \begin{cases} 1 \le l \le 3, \\ 1 \le i \le N, \\ 0 \le j \le g_{i,2}. \end{cases}$$

- Young animals that die and leave biomass in the form of meat and bones.

$$r_{21} \equiv [Y_{i,j}]_l^0 \overset{m_{i,1}}{\rightarrow} [H_i^{(f_{i,1}f_{i,5}+0.5)}, C_i^{(f_{i,2}f_{i,6}+0.5)}, B^{(f_{i,1}f_{i,5}+0.5)}, M^{(f_{i,2}f_{i,6}+0.5)}]_l^0, \quad \begin{cases} 1 \le l \le 3, \\ 0 \le j < g_{i,2}, \\ 1 \le i \le N. \end{cases}$$

- Young animals removed from the ecosystem that do not leave biomass.

$$r_{22} \equiv [Y_{i,j}]_l^0 \overset{m_{i,3}}{\rightarrow} [\quad]_l^0, \quad \begin{cases} 1 \le l \le 3, \\ 0 \le j < g_{i,2}, \\ 1 \le i \le N. \end{cases}$$

- Adult animals that survive.

$$r_{23} \equiv [Y_{i,j}]_l^0 \overset{1-m_{i,2}}{\rightarrow} [Z_{i,j}, D_i]_l^0, \quad \begin{cases} 1 \le l \le 3, \\ g_{i,2} \le j < g_{i,5}, \\ 1 \le i \le N. \end{cases}$$

- Adult animals that die and leave biomass.

$$r_{24} \equiv [Y_{i,j}]_l^0 \overset{m_{i,2}}{\rightarrow} [H_i^{(f_{i,3}f_{i,5}+0.5)}, C_i^{(f_{i,4}f_{i,6}+0.5)}, B^{(f_{i,3}f_{i,5}+0.5)}, M^{(f_{i,4}f_{i,6}+0.5)}]_l^0, \quad \begin{cases} 1 \le l \le 3, \\ g_{i,2} \le j \le g_{i,5}, \\ 1 \le i \le N. \end{cases}$$

- Animals that die by hunter and can leave biomass or not

$$r_{25} \equiv [Y_{i,j}]_l^0 \overset{m_{i,2}}{\rightarrow} [H_i^{(f_{i,3}f_{i,5}hp_i+0.5)}, C_i^{(f_{i,4}f_{i,6}hp_i+0.5)}, B_i^{(f_{i,3}f_{i,5}hp_i+0.5)}, M_i^{(f_{i,4}f_{i,6}hp_i+0.5)}]_m^0,$$

$$where\ 1 \le l \le 3, 1 \le i \le N, g_{i,2} \le j \le g_{i,5}.$$

- Randomnes generation of the total amount of animals. The following rules are applied at the same time than mortality rules.

$$r_{26} \equiv a'_{m,i}[\quad]_m^0 \rightarrow [a_i]_m^0, \quad \begin{cases} 1 \le m \le 3, \\ 1 \le i \le N. \end{cases}$$

$$r_{27} \equiv e'_{m,i}[\quad]_m^0 \overset{0.5}{\rightarrow} [a_i]_m^0, \quad \begin{cases} 1 \le m \le 3, \\ 1 \le i \le N. \end{cases}$$

$$r_{28} \equiv e'_{m,i}[\quad]_m^0 \overset{0.5}{\rightarrow} [\#]_m^0, \quad \begin{cases} 1 \le m \le 3, \\ 1 \le i \le N. \end{cases}$$

- Seasonal growth and production randomness.

$$r_{e5} \equiv (T'_{k,j}, R'_{k,s})_{e1}(\quad)_{ek} \rightarrow (\quad)_{e1}(N_{j,s})_{ek}, \quad \begin{cases} 1 \le j \le 100, \\ 1 \le s \le 100, \\ 1 \le k \le E. \end{cases}$$

$$r_{e6} \equiv (T'_{1,j}, R'_{1,s})_{e1} \to (N_{j,s})_{e1}, \quad \begin{cases} 1 \le j \le 100, \\ 1 \le s \le 100, \\ 1 \le k \le E. \end{cases}$$

Alliance production

- Objects $N'_{j,s}$ associated with altitude are introduced into the membrane to produce grass.

$$r_{29} \equiv N_{j,s}[\quad]_0^0 \to [N'_{1,j,s}, N'_{2,j,s}, N'_{3,j,s}]_0^0, \quad \begin{cases} 1 \le i \le 4, \\ 1 \le j \le 100, \\ 1 \le s \le 100. \end{cases}$$

$$r_{30} \equiv N'_{m,j,s}[\quad]_m^0 \to [N_{j,s}]_m^0, \quad \begin{cases} 1 \le m \le 3, \\ 1 \le j \le 100, \\ 1 \le s \le 100, \\ 1 \le k \le 4. \end{cases}$$

- Animal density control.

$$r_{31} \equiv [D_i^{(d_{i,1,m,k})}, a_i^{(d_{i,1,m,k}-d_{i,2,m,k})} \to \#]_m^0, \quad \begin{cases} 1 \le k \le E, \\ 1 \le i \le N, \\ 1 \le m \le 3. \end{cases}$$

- Grass production.

$$r_{32} \equiv [N'_{j,s}, A_i \to G_i^{(NZ_j\sigma T_i+\mu T_i)(NZ_j\sigma R_i+\mu R_i)}, A_i]_m^0, \quad \begin{cases} 1 \le m \le 3, \\ 1 \le j \le 100, \\ 1 \le s \le 100, \\ 1 \le k \le 4. \end{cases}$$

Feeding and density regulation

- When the animal $Z_{i,j}$ into the membrane m and environment k finds grass G_i and has enough space a_i it eat and evolve to $WN_{m,i,j}$ and abandons the membrane.

$$r_{33} \equiv [Z_{i,j}, a_i, G_k^{fa_i}]_m^0 \overset{ft_{i,k}}{\to} WN_{m,i,j}[\quad]_m^-, \quad \begin{cases} 1 \le j \le g_{i,6}, \\ 1 \le i \le N, \\ 1 \le k \le NA, \\ 1 \le m \le 3. \end{cases}$$

- The following rules generate the membrane charge change.

$$r_{34} \equiv [\rho_5]_m^0 \to [\rho_6]_m^-, \quad \{1 \le m \le 3.$$

$$r_{35} \equiv [\rho_6]_m^- \to [\rho_7]_m^0, \quad 1 \le m \le 3.$$

Change environment module

- The animals that don't eat ($Z_{i,j}$), grass production (G_i), the biomass deposited in the ecosystem (B_m and M_m) and the density regulator object a abandon the membrane m.

$$r_{36} \equiv [Z_{i,j}]_m^- \to Z'_{m,i,j}[\quad]_m^0, \quad \begin{cases} 0 \le j \le g_{i,6}, \\ 1 \le i \le N, \\ 1 \le m \le 3. \end{cases}$$

$$r_{37} \equiv [G_k]_m^- \to G'_{m,k}[\quad]_m^0, \quad \begin{cases} 1 \le k \le NA, \\ 1 \le m \le 3. \end{cases}$$

$$r_{38} \equiv [B]_m^- \to B'[\quad]_m^0, \quad 1 \le m \le 3.$$

$$r_{39} \equiv [M]_m^- \to M'[\quad]_m^0, \quad 1 \le m \le 3.$$

$$r_{40} \equiv [a_i]_m^- \to a'_{m,i}[\quad]_m^0, \quad \begin{cases} 1 \le i \le N, \\ 1 \le m \le 3. \end{cases}$$

- The same objects go out the skin membrane.

$$r_{41} \equiv [Z'_{m,i,j}]_0^0 \to Z'_{m,i,j}[\quad]_0^0, \quad \begin{cases} 1 \le j \le g_{i,6}, \\ 1 \le i \le N, \\ 1 \le k \le NA, \\ 1 \le m \le 3. \end{cases}$$

$$r_{42} \equiv [G'_{m,i}]_0^0 \to G'_{m,i}[\quad]_0^0, \quad \begin{cases} 1 \le i \le NA, \\ 1 \le m \le 3. \end{cases}$$

$$r_{43} \equiv [B'_m]_0^0 \to B'_m[\quad]_0^0, \quad 1 \le m \le 3.$$

$$r_{44} \equiv [M'_m]_0^0 \to M'_m[\quad]_0^0, \quad \{ 1 \le m \le 3.$$

$$r_{45} \equiv [a'_{m,i}]_0^0 \to a'_{m,i}[\quad]_0^0, \quad \begin{cases} 1 \le i \le N, \\ 1 \le m \le 3. \end{cases}$$

- Then the objects $Z''_{k,m,i,j}$, $G''_{k,m,i}$, $B''_{m,i}$, $M''_{m,i}$ and $a''_{k,m,i}$ abandon the environment k: $1 \le k \le 4$ and enter in environment 5.

$$r_{e7} \equiv (Z'_{m,i,j})_{ek}(\quad)_{e5} \to (\quad)_{ek}(Z''_{k,m,i,j})_{e5}, \quad \begin{cases} 1 \le i \le N, \\ 0 \le j \le g_{i,6}, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{e8} \equiv (G'_{m,i})_{ek}(\quad)_{e5} \to (\quad)_{ek}(G''_{k,m,i})_{e5}, \quad \begin{cases} 1 \le i \le NA, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{e9} \equiv (B'_m)_{ek}(\quad)_{e5} \to (\quad)_{ek}(B''_{k,m})_{e5}, \quad \begin{cases} 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{e10} \equiv (M'_m)_{ek}(\quad)_{e5} \to (\quad)_{ek}(M''_{k,m})_{e5}, \quad \begin{cases} 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{e11} \equiv (a'_{m,i})_{ek}(\quad)_{e5} \to (\quad)_{ek}(a''_{k,m,i})_{e5}, \quad \begin{cases} 1 \le i \le N, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{46} \equiv Z''_{k,m,i,j}[\quad]^0_0 \to [Z''_{k,m,i,j}]^0_0, \quad \begin{cases} 0 \le j \le g_{i,6}, \\ 1 \le i \le N, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{47} \equiv G''_{k,m,i}[\quad]^0_0 \to [G''_{k,m,i}]^0_0, \quad \begin{cases} 1 \le k \le E, \\ 1 \le i \le Na, \\ 1 \le m \le 3. \end{cases}$$

$$r_{48} \equiv B''_{k,m}[\quad]^0_0 \to [G''_{k,m}]^0_0, \quad \begin{cases} 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{49} \equiv M''_{k,m}[\quad]^0_0 \to [M''_{k,m}]^0_0, \quad \begin{cases} 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{50} \equiv a''_{k,m,i}[\quad]^0_0 \to [a''_{k,m,i}]^0_0, \quad \begin{cases} 1 \le k \le E, \\ 1 \le i \le N, \\ 1 \le m \le 3. \end{cases}$$

- These objects enter into a virtual environment ($e = 5$) and membrane $m = 4$.

$$r_{51} \equiv Z''_{k,m,i,j}[\quad]^0_4 \to [Z''_{k,m,i,j}]^0_4, \quad \begin{cases} 0 \le j \le g_{i,6}, \\ 1 \le i \le N, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{52} \equiv G''_{k,m,i}[\quad]^0_4 \to [G''_{k,m,i}]^0_4, \quad \begin{cases} 1 \le i \le Na, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{53} \equiv B''_{k,m}[\quad]^0_4 \to [B''_{k,m}]^0_4, \quad \begin{cases} 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{54} \equiv M''_{k,m}[\quad]^0_4 \to [M''_{k,m}]^0_4, \quad \begin{cases} 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{55} \equiv a''_{k,m,i}[\quad]^0_4 \to [a''_{k,m,i}]^0_4, \quad \begin{cases} 1 \le i \le N, \\ 1 \le k \le E, \\ 1 \le m \le 3. \end{cases}$$

Feeding rules

$$r_{56} \equiv [Z''_{k,m,i,j}, a''_{v,m,i}, G''^{fa_i}_{v,m,s}]^0_4 \overset{ft_{i,s} p_{i,k,v}}{\longrightarrow} W'_{v,m,i,j}[\quad]^-_4, \quad \begin{cases} 0 \le j \le g_{i,6}, \\ 1 \le i \le N, \\ 1 \le s \le Na, \\ 1 \le k \le E, \\ 1 \le v \le E, \\ 1 \le m \le 3. \end{cases}$$

$$r_{57} \equiv [\rho_{11}]^0_4 \to [\rho_{12}]^-_4$$

$r_{58} \equiv [\rho_{12}]_4^- \rightarrow [\rho_{13}]_4^0$

- When the charge of the membrane 5 changes to negative, the remaining $Z''k, m, i, j$ objects (aniamls that have not enough resources) transform to an object $V_{k,m,i,j}$ and it also abandons this membrane. The remaining objects $(G''_{k,m,i}, B''_{k,m}, M''_{k,m} and a''_{k,m,i})$ disappear. And the membrane change its polarity to null.

$$r_{59} \equiv [Z''_{k,m,i,j}]_4^- \rightarrow V_{k,m,i,j}[\quad]_4^0, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{60} \equiv [G''_{k,m,i}]_4^- \rightarrow [\#]_4^0, \quad \begin{cases} 1 \leq i \leq Na, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{61} \equiv [B''_{k,m}]_4^- \rightarrow [\#]_4^0, \quad \begin{cases} 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{62} \equiv [M''_{k,m}]_4^- \rightarrow [\#]_4^0, \quad \begin{cases} 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{63} \equiv [a''_{k,m,i}]_4^- \rightarrow [\#]_4^0, \quad \begin{cases} 1 \leq i \leq N, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

- The objects associated to the animals move to their environment.

$$r_{64} \equiv [W'_{k,m,i,j}]_0^0 \rightarrow W'_{k,m,i,j}[\quad]_0^0, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{65} \equiv [V_{k,m,i,j}]_0^0 \rightarrow V_{k,m,i,j}[\quad]_0^0, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{e12} \equiv (W'_{k,m,i,j})_{e5}(\quad)_{ek} \rightarrow (\quad)_{e5}(WN_{m,i,j})_{ek}, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{e13} \equiv (V_{k,m,i,j})_{e5}(\quad)_{ek} \rightarrow (\quad)_{e5}(V'_{m,i,j})_{ek}, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq k \leq E, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{66} \equiv WN_{m,i,j}[\quad]_0^0 \rightarrow [WN_{m,i,j}]_0^0, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

$$r_{67} \equiv V'_{m,i,j}[\quad]_0^0 \rightarrow [V'_{m,i,j}]_0^0, \quad \begin{cases} 0 \leq j \leq g_{i,6}, \\ 1 \leq i \leq N, \\ 1 \leq m \leq 3. \end{cases}$$

Updating module

$$r_{68} \equiv [P_{14} \rightarrow F_1, F_2, F_3, P_{15}]_0^0$$

$$r_{69} \equiv F_m[\quad]_m^0 \rightarrow [\quad]_m^-, \quad 1 \leq m \leq 3.$$

$$r_{70} \equiv WN_{m,i,j}[\quad]_m^- \rightarrow [X_{i,j+1}]_m^0, \quad \begin{cases} 1 \leq m \leq 3, \\ 1 \leq j < g_{i,5}, \\ 1 \leq i \leq N. \end{cases}$$

- The objects associated to animals that have not found resources, disappear leaving biomass.

$$r_{71} \equiv V'_{m,i,j}[\quad]_m^- \rightarrow [H_i^{(f_{i,1}f_{i,5}+0.5)}, C_i^{(f_{i,2}f_{i,6}+0.5)}, B_i^{(f_{i,1}f_{i,5}+0.5)}, M_i^{(f_{i,2}f_{i,6}+0.5)}]_m^0, \quad \begin{cases} 1 \leq m \leq 3, \\ 1 \leq j < g_{i,5}, \\ 1 \leq i \leq N. \end{cases}$$

$$r_{72} \equiv V'_{m,i,j}[\quad]_m^- \rightarrow [H_i^{(f_{i,3}f_{i,5}hp_i+0.5)}, C_i^{(f_{i,4}f_{i,6}hp_i+0.5)}, B_i^{(f_{i,3}f_{i,5}hp_i+0.5)}, M_i^{(f_{i,4}f_{i,6}hp_i+0.5)}]_m^0,$$
$$where\ 1 \leq m \leq 3, g_{i,2} \leq j < g_{i,5}, 1 \leq i \leq N.$$

- The objects associated to animals that have reached their life expectancy transform into objects associated to biomass.

$$r_{73} \equiv WN_{m,i,g_{i,5}}[\quad]_m^- \rightarrow [H_i^{(f_{i,3}f_{i,5}hp_i+0.5)}, C_i^{(f_{i,4}f_{i,6}hp_i+0.5)}, B_i^{(f_{i,3}f_{i,5}hp_i+0.5)}, M_i^{(f_{i,4}f_{i,6}hp_i+0.5)}]_m^0,$$
$$where\ 1 \leq m \leq 3, 1 \leq i \leq N.$$

$$r_{74} \equiv [P_{16}]_0^0 \rightarrow T, R[P_0]_0^0$$

$$r_{75} \equiv [\rho_{12}]_m^- \rightarrow [\rho_0]_m^0, \quad 1 \leq m \leq 3.$$

References

1. A. Aldezabal. *Análisis de la interacción vegetación-Grandes herbívoros en las comunidades supraforestales del Parque Nacional de Ordesa y Monte Perdido (Pirineo*

Central, Aragón). PhD thesis, Euskal Herriko Unibertsitatea/Universidad del Paś Vasco, Leioa, 1997.
2. A. Aldezabal, I. Garín, and R. García-González. Comparación de varios métodos para la estima de la producción primaria aérea en comunidades herbáceas subalpinas del pirineo central. *Actas de la XXXVI reunión científica de la SEEP.*, pages 167–171, 1996.
3. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M. Pérez-Jiménez, and D. Sanuy. A computational modeling for real ecosystems based on p systems. *Natural Computing*, 10:39–53, March 2010.
4. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M. Pérez-Jiménez, and D. Sanuy. A p system based model of an ecosystem of some scavenger birds. *Lecture Notes in Computer Science*, 5957:182–195, 2010.
5. M.A. Colomer, S. Lavín, I. Marco, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, E. Serrano, and L. Valencia-Cabrera. Modeling population growth of pyrenean chamois (rupicapra p. pyrenaica) by using p-systems. In M. Gheorghe, T. Hinze, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing. 11th International Conference, CMC 2010 Jena, Germany. Revised Selected Papers*, pages 144–159. Springer-Verlag, August 2010.
6. M.A. Colomer, A. Margalida, D. Sanuy, and 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*, 222:33–47, 2011.
7. Grup de Recerca de Geobotànica i Cartografia de la Vegetació. Geoveg. *Universitat de Barcelona*, 2011.
8. R. García-González, A. Marinas, D. Gómez, and A. Aldezabal. Revisión bibliográfica de la producción primaria neta aérea de las principales comunidades pascícolas pirenaicas. *XLII Reunión Científica de la SEEP*, 2001.
9. D. Gómez. *Pastos del Pirineo*, chapter Métodos para el estudio de los pastos, su caracterización ecológica y valoración. Dpto. de Publicaciones del CSIC, 2008.
10. C. Koerner. *Alpine Plant Life. Functional Plant Ecology of High Mountain Ecosystems*. Springer - Verlag, 1999.
11. A. Marinas and R. García-González. *Pastos del Pirineo*, chapter Aspectos productivos de los pastos pirenaicos. Dpto. de Publicaciones del CSIC, 2008.
12. Generalitat de Catalunya. Reserves Nacionals de Caa Natural environment department. http://www20.gencat.cat/portal/site/dmah.
13. Ignacio Pérez-Hurtado, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez, M. A. Colomer, and Agustín Riscos-Núñez. 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)*, 1:637–643, 2010.
14. S. Rivas-Martínez, T.E. Díaz, F. Fernández-González, J. Izco, J. Loidi, M. Lousa, and A. Penas. Vascular plant communities of spain and portugal. addenda to the syntaxonomical checklist of 2001. *Itinera Geobotanica*, 15:5–922, 2002.
15. J.S. Singh, W.K. Lauenroth, and R.K. Steinhorst. Review and assessment of various techniques for estimating net aerial primary production in grasslands from harvest data. *The Botanical Review*, 41(2):181–232, 1975.

P Systems for Social Networks

Erzsébet Csuhaj-Varjú[1], Marian Gheorghe[2], György Vaszil[1], Marion Oswald[3]

[1] Computer and Automation Research Institute
Hungarian Academy of Sciences
H-1111, Budapest, Kende u. 13-17, Hungary
{csuhaj,marion,vaszil}@sztaki.hu
[2] Department of Computer Science
University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, United Kingdom
m.gheorghe@dcs.shef.ac.uk
[3] Institute for Computer Languages
Vienna University of Technology
Favoritenstr. 9-11, 1040 Vienna, Austria
marion@logic.at

Summary. We introduce some variants of P systems that mimic the behaviour of social networks and illustrate some of the characteristics of them. Other concepts related to social networks are discussed and suitable classes of P systems are suggested. A simple example shows the capabilities of such a P system where the intensity of the communication is modelled with complementary alphabets.

1 Introduction

Membrane computing (also called P systems theory) is a new computing paradigm, which in its initial variants was inspired by the structure and functionality of the living cell [19]. Later on some other biological entities have been considered in order to extend the capabilities of this computational model - tissues or special types of cells, like neurons, or colonies of cells, like bacteria. So far, concepts and methods of P systems theory have been successfully employed in solving important problems of computer science and describing various biological phenomena, but, except promising applications in linguistics and natural language processing, only a limited amount of attention has been paid to the suitability of membrane systems in modelling social phenomena.

In this paper, we attempt to build a bridge between membrane computing and the theory of *social networks*, an area of great interest in contemporary computer science and practice. For this reason, we define certain classes of P systems which are suitable for modelling features of social networks and which can be derived from problems in this field, and we formulate various research topics related to

connections between P systems theory and the theory of social interactions and networks. The underlying mathematical tool set is the theory of formal languages, i.e., our approach to social networks is a purely syntactic one. We note that social phenomena have already been described by different frameworks in formal language theory, our recent aim is to formulate models which combine tools of both membrane computing and formal language theory. In terms of formal grammars, communities of agents interacting with each other and with their dynamically changing environment were modelled by eco-grammar systems, a research field launched in [9]. Population of agents, called networks of (parallel) language processors, with biological and social background were described by rewriting systems in [12, 7]. Another formal language-theoretic model for communities of evolving agents, called evolutionary systems, was introduced in [10]. Multi-agent systems in terms of formal language theory and membrane computing were discussed in [4], [5], [3], and [15].

2 Social Networks

In various formalisms related to the study of social phenomena, interpersonal relationships between individuals are defined as information-carrying connections [14]. These relationships come in various forms, two of them are *strong* and *weak ties*. Weak ties seem to be responsible for the embeddedness and structure of social networks and for the communication within these systems [14]. There are other measures that characterise connections between nodes (individuals). *Centrality* gives an indication of the social power of a node and the strength of its connections. It relies on other measures, *betweenness, closeness, degree*. Betweenness measures to what extent a node is connected to nodes that have a significant number of neighbours (direct connections). Closeness is the degree describing that a node is close to all other nodes in the network: it counts the number of connections. For the above concepts as well as for other measures of the connections existing in social networks, we refer to [23].

3 P Systems Capturing Communication Aspects

We are focusing now on identifying some classes of P systems that capture communication aspects in social networks. We can consider various types of nodes: *ordinary* or *popular* nodes - those that host individuals and allow communication between them; *new-born* nodes - those that are dynamically created and linked to the existing network; *non-visible* or *extinct* nodes - the nodes that are no longer connected to the network or have disappeared; nodes with one way communication, only allowing information to go into, *blackholes* or allowing only to exit from, *whiteholes*. Some of these aspects have been already considered in the current research framework of membrane computing; for instance population P systems allow nodes

to be dynamically connected and disconnected [6]; the one-way communication for communicating accepting P systems has been considered in [13]. We can also take into account connections between nodes and look at the *volume of communication* - the amount of (new) information generated or sent-received by various nodes or groups of nodes; *frequency of communicated messages* - the number of communication steps related to the evolution (computation) steps; *communication motifs* - patterns of communication identified throughout the network evolution. In order to capture these phenomena we aim to formally define a generic and flexible framework whereby these concepts can be appropriately accommodated. In this respect we provide the following general definition of a *population P system governed by communication*, a *pgcP system*, for short.

4 Preliminaries and Definitions

Throughout the paper we assume that the reader is familiar with the basics of membrane computing and formal language theory; for details we refer to [17, 21] and [22]. For an alphabet V, we denote by $|V|$ the cardinality of V, and by V^* the set of all finite words over V. If λ, the empty word is excluded, then we use the notation V^+. As usual in membrane computing, we represent the finite multisets over V by strings over V as well, that is, a string and all its permutations correspond to the same multiset. We denote by $|w|_a$ the number of occurrences of a symbol $a \in V$ in a string $w \in V^*$ which is equal to the multiplicity of that object in the represented multiset. We use $\emptyset$ to denote the empty multiset, and also use V^* to denote the set of finite multisets over an alphabet V.

In the following we consider P systems with static underlying graph structure; the notion can easily be extended to a construct capturing dynamically changing underlying graph architecture as well.

Definition 1. *A population P system governed by communication (a pgcP system, for short), is a construct*

$$(\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_k, R_k)),\ n, k \geq 1,$$

where

- $\Sigma = \Sigma_1 \cup \Sigma_2$ *is a finite alphabet of objects,* Σ_1 *is the alphabet of cellular objects, i.e., the objects in the nodes, and* Σ_2 *is the set of communication symbols;*
- $E \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$ *is the set of (directed) links between the nodes;*
- $\omega_i \in \Sigma_1^*,\ 1 \leq i \leq n$, *is a multiset of cellular objects, the initial content of the node* i *of the system; and*
- $(\rho_i, R_i),\ 1 \leq i \leq k$, *are predicate based rule-sets governing the transitions of the system, with*
  - $\rho_i : \Sigma_2^* \rightarrow \{true, false\}$, *a predicate over the multisets of communication symbols, and*

– $R_i = (R_{i,1}, \ldots, R_{i,n})$, *an* n*-tuple of sets of rewriting rules, where* $R_{i,j}$, $1 \leq j \leq n$, *is the set of rules that are allowed to be applied at node* j, *and any rule is of the form* $u \to v$ *for* $u \in \Sigma_1^*$, $v \in (\Sigma_1 \cup (\Sigma_1 \times \Sigma_2 \times Tar))^*$ *where* $Tar = \{1, \ldots n\}$ *is a set of target indicators.*

Definition 2. *A configuration of a pgcP system*

$$\Pi = (\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_k, R_k)),\ n, k \geq 1,$$

is an $n + s$*-tuple for* $s = |E|$,

$$(w_1, ..., w_n; u_1, ..., u_s),\ w_i \in \Sigma_1^*,\ u_j \in \Sigma_2^*,\ 1 \leq i \leq n,\ 1 \leq j \leq s,$$

where multiset w_i *is the multiset of cellular objects at the* i*-th node, i.e., the current content of node* i, $1 \leq i \leq n$, *and* u_j *is the multiset of communication symbols associated to the communication link* $e_j \in E$, $1 \leq j \leq s$.

The initial configuration of Π *is* $(\omega_1, \ldots, \omega_n; \emptyset, \ldots, \emptyset)$.

The pgcP system works by changing its configurations. In the following we describe the transition or configuration change: it takes place by rewriting and communication of the cellular objects and recording the performed communication. The rewriting rules are applied to the cellular objects in the maximally parallel manner, i.e., any object can be involved in at most one rule application, and as many rules are applied simultaneously to the cellular objects at the nodes as possible.

Definition 3. *Let* $\Pi = (\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_k, R_k))$, $n, k \geq 1$, *be a pgcP system and let* $c_1 = (w_1, ..., w_n; u_1, ..., u_s)$ *and* $c_2 = (w'_1, ..., w'_n; u'_1, ..., u'_s)$ *be two configurations of* Π.

We say that c_1 *changes directly to* c_2 *(or* c_2 *is obtained from* c_1 *with a transition), denoted by*

$$c_1 = (w_1, ..., w_n; u_1, ..., u_s) \Rightarrow^{(\rho_i, R_i)} c_2 = (w'_1, ..., w'_n; u'_1, ..., u'_s)$$

for some $i \in \{1, \ldots, k\}$, *if the following hold:*

- $\rho_i(u_1, \ldots, u_s) = true$,
- c_1 *is changed to* c_2 *by using the rules of* $R_i = (R_{i,1}, \ldots, R_{i,n})$ *as follows:*
  - *the rules of* $R_{i,j}$ *are applied in the maximal parallel manner in the node* j *to the multiset* w_j, $1 \leq j \leq n$; *and*
  - *if a rule of the form* $u \to v$ *where* $v = v_1 v_2$ *for* $v_1 \in \Sigma_1^*$ *and* $v_2 \in (\Sigma_1 \times \Sigma_2 \times Tar)^*$ *is applied in a node* j, *then the following holds:*
    - u *is changed to* $v_1 v_2$ *and all objects in* v_1 *remain in node* j;
    - *all symbols of* $(a, a', l) \in v_2$ *are processed by sending the cellular object* $a \in \Sigma_1$ *to node* l *and adding the communication object* $a' \in \Sigma_2$ *to the multiset associated to the link* $(j, l) \in E$, $1 \leq j \neq l \leq n$.

The pgcP system may record information on the communication performed during the whole computation or only on communication during the last configuration change. This is captured in the following definition.

Definition 4. *Let $\Pi = (\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_k, R_k))$, $n, k \geq 1$, be a pgcP system. We say that Π works in the*

- *history preserving mode, if for any transition*

$$(w_1, ..., w_n; u_1, ..., u_s) \Rightarrow^{(\rho_i, R_i)} (w'_1, ..., w'_n; u'_1, ..., u'_s)$$

  it holds that $u'_j = u_j u''_j$, for $1 \leq j \leq s$, where u''_j is the multiset of communication symbols sent to link j during the transition,
- *non-history preserving mode, if u'_j consists of the communication symbols sent to link j during the transition (the communication symbols in u_j are forgotten).*

Definition 5. *Let $\Pi = (\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_k, R_k))$, $n, k \geq 1$, be a pgcP system. A derivation (or computation) in Π is a sequence of transitions starting in the initial configuration and ending in some final (possibly halting) configuration.*

The *result* of a computation in a pgcP system Π can be defined in various manners. We may consider the number (vector) of (certain) *communication objects* going through (certain) communication links or the number (vector) of (certain) *cellular objects* in (certain) nodes. If we assume distinguished, so called output link(s) or node(s), then we indicate this fact in the notation for the accepted languages of the pgcP system. As usual in membrane computing we consider only halting computations.

Definition 6. *Let $\Pi = (\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_k, R_k))$, where $n, k \geq 1$, and let $T_i \subseteq \Sigma_i$, $1 \leq i \leq 2$, be the sets of terminal cellular and communication objects; $Out_1 \subseteq \{1, \ldots, n\}$ and $Out_2 \subseteq E$ be the sets of output nodes and output links, respectively.*

- *We define $N_{cell}(\Pi, T_1, Out_1)$ ($Ps_{cell}(\Pi, T_1, Out_1)$) as the number (vector) of terminal cellular objects in the output nodes in a final configuration.*
- *We define $N_{com}(\Pi, T_2, Out_2)$ ($Ps_{com}(\Pi, T_2, Out_2)$) as the number (vector) of terminal communication objects associated to the output links in a final configuration.*

5 Complementary Alphabets in pgcP Systems

We are focusing now on communication in these networks. In order to characterize its intensity and the fact that the importance of a connection might evolve in time by either increasing or decreasing its value, we would need some sort of symbols that act in this respect. One way to model this is by considering complementary

communication symbols, whereby the customary (or positive) symbols strengthen a connection, whereas the complementary (negative) ones weaken it. Formally this is achieved by splitting the alphabets Σ_1 and Σ_2 as follows:

$$\Sigma_i = \Sigma_i' \cup \bar{\Sigma}_i' \cup \Sigma_i''$$

where Σ_i' and $\bar{\Sigma}_i'$, $i = 1, 2$ are *dual alphabets*. Σ_i' consists of normal (positive) elements and $\bar{\Sigma}_i'$ contains complementary symbols.

The idea of complementary alphabets is not new in the field of natural computing. *DNA computing* has as a core data structure a double-strand structure consisting of dual elements, the DNA nucleotides, adenine, thymine, cytosine and guanine represented by the four letter alphabet, $\{A, T, C, G\}$, respectively; the pairs (A, T) and (C, G) are known as *complementary base pairs* [20]. In the context of *networks of Watson-Crick D0L systems*, networks of such systems over DNA-like alphabets are introduced and operations relying on complementarity properties are utilised [8, 11].

Active and passive objects have been considered in a similar way with complementary elements. Two types of such objects have been introduced and studied: within components [1] and on membranes [2]. Other membrane systems using complementary features are *spiking neural P systems with anti-spikes* where a neuron receiving s spikes and t anti-spikes is left with $s - t$ if $s \geq t$ or $t - s$ when $t \geq s$ objects [18], and membrane systems with bi-stable catalysts, where the system may switch between two states [17].

To demonstrate the above ideas, we present a simulation of an n-register machine M where the communication is controlled by "Watson-Crick-like" predicates.

An n-register machine $M = (Q, R, q_0, q_f, P)$, $n \geq 1$, is defined as usual, that is, with internal state set Q, registers $R = (A_1, \ldots, A_n)$, initial and final states $q_0, q_f \in Q$, respectively, and a set of instructions P of the form (q, A_i+, r, s) or (q, A_i-, r, s), $q, r, s \in Q$, $q \neq q_f$, $A_i \in R$. When an instruction of the first type is performed, then M is in state q, increases the value of register A_i by one, and enters a state r or s, chosen nondeterministically. When an instruction of the second type is performed, then M is in state q and it subtracts one from the value of register A_i if it stores a positive number and then enters state r, or it leaves the value of A_i unchanged if it stores zero and then enters state s. There are no instructions for the final state, therefore the machine halts after entering state q_f. M starts its work in the initial state, q_0, with empty registers. Then it performs a sequence of instructions; if the sequence is finite (it ends with halting), then we speak of a computation by M. The result of the computation is the number stored in the output register A_1 after halting.

It is known that 2-register machines are able to compute any recursively enumerable set of numbers [16].

Example 1. Let $M = (Q, R, q_0, q_f, P)$, $n \geq 1$, be an n-register machine. We construct the pgcP system Π with $n + 1$ nodes simulating M as follows. Let

$$\Pi = (\Sigma, T_1, E, w_0, w_1, \ldots, w_n, (\rho_1, R_1), (\rho_2, R_2), Out)$$

where $\Sigma = \Sigma_1 \cup \Sigma_2$ with $\Sigma_1 = \{a, \bar{a}\} \cup \{q, [q, r, s] \mid q, r, s \in Q\}$, $\Sigma_2 = \{a, \bar{a}\}$, and $E = \{(0, i) \mid 1 \leq i \leq n\}$, $T_1 = \{a\}$, and $Out = 1$. The initial configuration and the rule sets are defined as follows. Let

$$w_0 = q_0, \text{ and } w_i = \emptyset, \ 1 \leq i \leq n.$$

Moreover, let $R_1 = (R_{1,0}, \ldots, R_{1,n})$, $R_2 = (R_{2,0}, \ldots, R_{2,n})$, and let

$$\begin{aligned} \rho_1(u_1, \ldots, u_n) \ : \ & |u_i|_a \geq |u_i|_{\bar{a}} \text{ for all } 1 \leq i \leq n, \\ R_{1,0} = \ & \{q \to [q, r, s](a, a, j) \mid (q, A_j+, r, s) \in P\} \cup \\ & \{q \to [q, r, s](\bar{a}, \bar{a}, j) \mid (q, A_j-, r, s) \in P\} \cup \\ & \{[q, r, s] \to r, [q, r, s] \to s \mid q, r, s \in Q\}, \end{aligned}$$

$$\begin{aligned} \rho_2(u_1, \ldots, u_n) \ : \ & |u_i|_a < |u_i|_{\bar{a}} \text{ for some } i, \ 1 \leq i \leq n, \\ R_{2,0} = \ & \{[q, r, s] \to s(a, a, j) \mid (q, A_j-, r, s) \in P\}, \text{ and finally} \end{aligned}$$

$$R_{j,k} = \{a\bar{a} \to \varepsilon\} \text{ for all } 1 \leq j \leq 2, \ 1 \leq k \leq n.$$

If Π works in the history preserving mode, then the difference between the number of symbols a and $\bar{a}$ on a link $(0, j)$ for some $1 \leq j \leq n$ corresponds to the value of register j.

The result of the computation can be found in node 1 corresponding to the output register A_1 of M if the system introduces the symbol q_f in node 0 and halts (because there is no rule for the final state in M). Thus, these variants of pgcP systems are computationally complete.

The reader may observe that in the above example we have not only complementary alphabets, but through various global predicates, the sets of rules are split into "normal" rules in R_1 and "recovery" rules in R_2.

We note that it is possible to simulate a register machine with one active component and n others receiving values a or $\bar{a}$. In a very similar way, we can consider a distributed model where the n components corresponding to registers are used in such a way that each one has its own addition and subtraction rules, similar to $R_{1,0}, R_{2,0}$. In this case we have to communicate not only a or $\bar{a}$ but also the label of the next register to the component simulating this register; some other variants of rules can be considered. Regarding these two models, one immediate question, perhaps not difficult to be addressed, is which one is simpler, more efficient - with respect to the number of rules, symbols etc; or are they just the same? What about simulating one with the other one?

Furthermore, the example of simulating a register machine suggests the use of *dual sets of rules*, triggered by predicates, i.e., to have for each rule, r, its dual rule, $\bar{r}$, defined in such a way that the complementary rules introduce complementary symbols (only).

6 Further Research Topics

In the previous sections we introduced the concept of a pgcP system inspired by social networks and made some steps towards identifying the necessary abstract elements related to measuring the intensity of the communication in these systems. In this respect, complementary alphabets and particular variants of pgcP systems have been considered. In the following, we define some further concepts regarding some specific types of pgcP systems, and list some preliminary results for these so-called *deterministic* and *non-cooperative* pgcP systems.

Definition 7. *Let* $\Pi = (\Sigma, E, \omega_1, \ldots, \omega_n, (\rho_1, R_1), \ldots, (\rho_s, R_s))$, $n, s \geq 1$, *be a non-cooperative deterministic pgcP system.*

Let $c(t) = (w_1(t), ..., w_n(t); u_1(t), ..., u_s(t))$, $t \geq 0$, *be a computation in* Π *in the history preserving mode.*

We define the

1. *growth of communication volume on link* i *at derivation step* t, $t \geq 1$, *by* $f_i : \mathbb{N} \to \mathbb{N}$ *where* $f_i(t) = |u_i(t)| - |u_i(t-1)|$.
2. *frequency of communication on link* i*:* $h_i : \mathbb{N} \to \{0, 1\}$ *where* $h_i(t) = 0$ *if* $|u_i(t)| - |u_i(t-1)| = 0$ *and* $h_i(t) = 1$ *if* $|u_i(t)| - |u_i(t-1)| \geq 1$*;*
3. *intensity of communication on link* i*:* $g_i : \mathbb{N} \to \mathbb{R}$, *where* $g_i(t) = \frac{f_i(t)}{t}$.

If $\Sigma_2 = \Sigma_2' \cup \bar{\Sigma}_2' \cup \Sigma_2''$, i.e., complementary symbols are considered, then $\bar{f}_i(t)$ defined over $\bar{\Sigma}_2'$, and difference functions as $f_i(t) - \bar{f}_i(t)$ can also be defined and examined.

We note that the above concepts can be extended with suitable modifications to gcpP systems in the general sense; obviously, in this case we speak of relations, instead of functions.

These notions have their roots in concepts related to networks of parallel language processors [12] and evolutionary systems [10]. A network of parallel language processors with D0L systems as components (an NLP-D0L system) consists of D0L systems located in nodes of a finite virtual graph (each node has at most one D0L system) which rewrite and communicate multisets of strings present in the nodes according to their own rule sets. A D0L system $G = (V, P, \omega)$ is a triplet, where V is an alphabet, P is a finite set of rules of the form $a \to \alpha$ with $a \in V$, $\alpha \in V^*$ and for each $a \in V$ there exists exactly one rule in P, and, finally, $\omega \in V^+$. For any string $x = x_1 \ldots x_n$, $x_i \in V$, $1 \leq i \leq n$, we say that x directly derives $y = y_1 \ldots y_n$, if $x_i \to y_i \in P$ holds for $1 \leq i \leq n$. (For more details on D0L systems, we refer to [21]). The NLP-D0L system functions with alternating rewriting and communication steps. By rewriting, each string at every node is rewritten in parallel; the D0L systems work in a synchronized manner. By communication, a copy of each string at a node is sent to each other node, given that the string satisfies the sender node's output context condition (predicate) and the receiver node's input context condition (predicate). Communication is performed in a parallel and synchronized

manner as well. In [12] it was shown that if the conditions for communication are random context conditions, i.e., they check the presence and /or the absence of certain symbols in the strings to be communicated, then the growth of the number of the strings in the network can be described by a growth function of a D0L system. The growth function of a D0L system orders to the number of derivation steps the length of the string obtained at that step. It was also shown, that the number of strings at specific nodes and the number of communicated strings between nodes can also be obtained from $D0L$ growth functions with suitable homomorphisms. The idea of the proofs comes from the property that in the case of $D0L$ systems any string generates only one string and the alphabet of the successor string can be calculated from the alphabet of the predecessor string.

The reader may easily notice the close relation between NLP-D0L systems and non-cooperative deterministic pgcP systems: The multisets of different symbols communicated from a node to some other one by an NLP-D0L system at any computation step corresponds to multisets of communication symbols added to the links of an appropriate pgcP system (where the predicates checks the presence/absence of types of objects in the multiset). Therefore, we may describe the growth of the communication volume, the frequency, and the intensity of communication on the links by tools of Lindenmayer systems, in particular the theory of D0L systems. Since D0L systems demonstrate several nice decidability properties, the theory provides efficient tools for characterizing the behaviour of particular types of pgcP systems. The detailed comparison is a topic for future research.

In context of social networks and pgcP systems, a number of other general problems can also be formulated. For example, how to describe and characterize other *concepts and measures* from social networks and how to define and model problems like *leaders and clusters emergence*. Or, how to dynamically restructure the links and distinguish between *good and bad or strong and weak links*; what about *breaking the links*. Finally, how to *solve various problems or compute functions* with such systems. These and similar questions form the basis of challenging future research.

7 Acknowledgement

The work of Erzsébet Csuhaj-Varjú and György Vaszil was supported in part by the Hungarian Scientific Research Fund, OTKA, Grant no. K75952. The work of Marian Gheorghe was partially done during his visit to the Computer and Automation Research Institute, Hungarian Academy of Sciences, in June 2010, and partially was supported by the grant K75952, Hungarian Scientific Research Fund, OTKA.

References

1. B. Aman, G. Ciobanu. Turing completeness using three mobile membranes. In *Unconventional Computing 2009*, LNCS, 5715, 42–55, 2009.
2. B. Aman, G. Ciobanu. Mutual mobile membranes systems with surface objects. In *7-th Brainstorming Week of Membrane Computing*, 29–39, 2009.
3. G. Bel-Enguix. A Multi-agent Model for Simulating the Impact of Social Structure in Linguistic Convergence. In*ICAART(2)* (J. Filipe et. al, Eds.), INSTICC Press, 367–372, 2009.
4. G. Bel-Enguix, M. A. Grando, M. D. Jiménez López. A Grammatical Framework for Modelling Multi-Agent Dialogues. In *PRIMA 2006* (Z.-Z. Shi, R. Sadananda, Eds.), LNAI 4088, Springer Verlag, Berlin Heidelberg, 10–21, 2009.
5. G. Bel-Enguix, M. D. Jiménez López. Membranes as Multi-agent Systems: an Application for Dialogue Modelling. In *IFIP PPAI 2006* (J.K. Debenham, Ed.), Springer, 31–40, 2006.
6. F. Bernardini, M. Gheorghe. Population P systems. *Intern J of Universal Comp Sci*, 10, 509–539, 2004.
7. E. Csuhaj-Varjú. Networks of Language Processors. *EATCS Bulletin* 63, 120–134, 1997.
8. E. Csuhaj-Varjú. Computing by networks of Watson-Crick *D0L* systems. In *Proc. Algebraic Systems, Formal Languages and Computation* (M. Ito, Ed.) RIMS Kokyuroku 1166, August 2000, Research Institute for Mathematical Sciences, Kyoto University, Kyoto, 43–51, 2000.
9. E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun. Eco-Grammar Systems: A Grammatical Framework for Studying Lifelike Interactions. *Artificial Life*, 3(3), 1–28, 1997.
10. E. Csuhaj-Varjú, V. Mitrana. Evolutionary systems: a language generating device inspired by evolving communities of cells. *Acta Informatica*, 36(11), 913–926, 2000.
11. E. Csuhaj-Varjú, A. Salomaa. Networks of Watson-Crick *D0L* systems. In *Words, Languages & Combinatorics III. Proceedings of the International Colloquium, Kyoto, Japan, March 14-21, 2000.* (M. Ito, T. Imaoka, Eds.), World Scientific Publishing Co., Singapore, 134–149, 2003.
12. E. Csuhaj-Varjú, A. Salomaa. Networks of Parallel Language Processors. In *New Trends in Formal Languages. Control, Cooperation, and Combinatorics* (Gh. Păun, A. Salomaa, Eds.), LNCS 1218, Springer Verlag, Berlin Heidelberg, 299-318, 1997.
13. E. Csuhaj-Varjú, G. Vaszil. P automata or purely communicating accepting P systems. In *Membrane Computing*(Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), LNCS 2597, Springer Verlag, Berlin Heidelberg, 219–233, 2003.
14. M.D. Granovetter. The Impact of Social Structures on Economic Development. *Journal of Economic Perspectives*, 19, 33–50, 2004.
15. M. D. Jiménez López. Agents in Formal Language Theory: An Overview. In *Highlights in Practical Applications of Agents and Multiagent Systems. 9th International Conference on Practical Applications of Agents and Multiagent Systems* (J. Bajo Pérez et. al, Eds.) Advances in Intelligent and Soft Computing 89, Springer, 283–290, 2011.
16. M. Minsky. *Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, 1967.
17. Gh. Păun. *Membrane Computing. An Introduction.* Springer, 2002.

18. L. Pan, Gh. Păun. Spiking neural P systems with anti-spikes, *Int J Computers Comms Control*, 4, 273–282, 2009.
19. Gh. Păun. Computing with Membranes. *J. of Comput. Syst. Sci.*, 61, 108–143, 2000.
20. Gh. Păun, G. Rozenberg, A. Salomaa. *DNA Computing - New Computing Paradigms*. Springer Verlag, 1998.
21. G. Rozenberg, A. Salomaa. (Eds). *Handbook of Formal Languages I-III*. Springer, 1997.
22. Gh. Păun, G. Rozenberg, A. Salomaa. (Eds). *The Handbook of Membrane Computing*. Oxford University Press, 2009.
23. S. Wasserman, K. Faust. *Social Networks Analysis: Methods and Applications*. Cambridge University Press, 1994.

Using Central Nodes to Improve P System Synchronization

Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
{mjd,yun,radu}@cs.auckland.ac.nz

Summary. We present an improved solution for the Firing Squad Synchronization Problem (FSSP) for digraph-based P systems. We improve our previous FSSP algorithm by allowing the general to delegate a more central cell in the P system to send the final command to synchronize. With e being the eccentricity of the general and r denoting the radius of the underlying digraph, our new algorithm guarantees to synchronize all cells of the system, between $e + 2r + 3$ steps (for all trees structures and many digraphs) and up to $3e + 7$ steps, in the worst case for any digraph. Empirical results show our new algorithm for tree-based P systems yields at least 20% reduction in the number of steps needed to synchronize over the previous best-known algorithm.

1 Introduction

The *Firing Squad Synchronization Problem* (FSSP) is one of the best studied problems for cellular automata, originally proposed by Myhill in 1957 [11]. The initial problem involves finding a cellular automaton, such that after the "firing" order is given by the general, after some finite time, all the cells in a line enter a designated *firing* state, *simultaneously* and *for the first time*. For an array of length n with the general at one end, minimal time $(2n - 2)$ solutions was presented by Goto [6], Waksman [18] and Balzer [2]. Several variations of the FSSP have been proposed and studied [12, 15]. The FSSP have been proposed and studied for variety of structures [10, 13, 7, 4].

In the field of membrane computing, deterministic solutions to the FSSP for a tree-based P system have been presented by Bernardini et al. [3] and Alhazov et al. [1]. For digraph-based P systems, we presented a deterministic solution in [5] for the generalized FSSP (in which the general is located at an arbitrary cell of the digraph), which runs in $3e + 11$ steps, where e is the eccentricity of the general.

In this paper, we present an improved FSSP solution for tree-based P systems, where the key improvement comes in having the general delegate a more central cell, as an alternative to itself, to broadcast the final "firing" order, to enter the

firing state. We also give details on how to use this approach to improve the synchronization time of digraph-based P systems.

It is well known in cellular automata [17], where "signals" with propagating speeds 1/1 and 1/3 are used to find a half point of one-dimensional arrays; the signal with speed 1/1 is reflected and meets the signal with speed 1/3 at half point. We generalize the idea used in cellular automata to find the center of a tree that defines the membrane structure of a P system.

Let r denote the radius of the underlying graph of a digraph, where $e/2 \leq r \leq e$. Our new algorithm is guaranteed to synchronize in t steps, where $e + 2r + 3 \leq t \leq 3e + 7$. In fact, the lower bound is achieved, for all digraphs that are trees. In addition to our FSSP solution, determining a center cell has many potential real world applications, such as *facility location problems* and *broadcasting*.

The rest of the paper is organized as follows. In Section 2, we give some basic preliminary definitions including our P system model and formally introduce the synchronization problem that we solve. In Section 3, we provide a detailed P system specification for solving the FSSP for tree-based P systems. In Section 4, we provide a detailed P system specification for solving the FSSP for digraph-based P systems. Finally, in Section 5, we summarize our results and conclude with some open problems.

2 Preliminary

We assume that the reader is familiar with the basic terminology and notations, such as relations, graphs, nodes (vertices), edges, directed graphs (digraphs), directed acyclic graphs (dag), arcs, alphabets, strings and multisets.

For a digraph (X, δ), recall that $\texttt{Neighbor}(x) = \delta(x) \cup \delta^{-1}(x)$. The relation $\texttt{Neighbor}$ is always symmetric and defines a graph structure, which will be here called the virtual *communication graph* defined by δ.

A special node $g \in X$ is designated as the *general*. For a given general g, we define the *depth* of a node x, $\texttt{depth}_g(x) \in \mathbb{N}$, as the length of a shortest path between g and x, over the $\texttt{Neighbor}$ relation. Recall that the *eccentricity* of a node $x \in X$, $\texttt{ecc}(x)$, as the maximum length of a shortest path between x and any other node. We note $\texttt{ecc}(g) = \max\{\texttt{depth}_g(x) \mid x \in X\}$.

Recall that a (free or unrooted) tree has either one or two *center nodes*—any node with minimum eccentricity. We denote a tree $T = (X, A)$, rooted at node $g \in X$ by T_g. The height of a node x in T_g is denoted by $\texttt{height}_g(x)$. For a tree T_g, we define the *middle node* to be the center node closest to g of the underlying tree T of T_g. Let $T_g(x)$ denote the subtree rooted at node x in T_g.

Given nodes x and y, if $y \in \texttt{Neighbor}(x)$ and $\texttt{depth}_g(y) = \texttt{depth}_g(x) + 1$, then x is a *predecessor* of y and y is a *successor* of x. Similarly, a node z is a *peer* of x, if $z \in \texttt{Neighbor}(x)$ and $\texttt{depth}_g(z) = \texttt{depth}_g(x)$. Note that, for node x, the set of peers and the set of successors are disjoint with respect to g. For node x, $\texttt{Pred}_g(x) = \{y \mid y \text{ is a predecessor of } x\}$, $\texttt{Peer}_g(x) = \{y \mid y \text{ is a peer of } x\}$ and $\texttt{Succ}_g(x) = \{y \mid y \text{ is a successor of } x\}$.

Definition 1. A *P system* of order n with *duplex* channels and *cell states* is a system $\Pi = (O, K, \delta)$, where:

1. O is a finite non-empty alphabet of *objects*;
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a finite set of *cells*;
3. δ is an *irreflexive* binary relation on K, which represents a set of structural arcs between cells, with duplex communication capabilities.

Each cell, $\sigma_i \in K$, has the initial configuration $\sigma_i = (Q_i, s_{i0}, w_{i0}, R_i)$, and the current configuration $\sigma_i = (Q_i, s_i, w_i, R_i)$, where:

- Q_i is a finite set of *states*;
- $s_{i0} \in Q_i$ is the *initial state*; $s_i \in Q_i$ is the *current state*;
- $w_{i0} \in O^*$ is the *initial content*; $w_i \in O^*$ is the *current content*; note that, for $o \in O$, $|w_i|_o$ denotes the *multiplicity* of object o in the multiset w_i;
- R_i is a finite *ordered* set of multiset rewriting *rules* (with *promoters*) of the form: $s\ x \rightarrow_\alpha s'\ x'\ (u)_\beta \mid z$, where $s, s' \in Q$, $x, x', u \in O^*$, $z \in O^*$ is the promoter [9], $\alpha \in \{\texttt{min}, \texttt{max}\}$ and $\beta \in \{\uparrow, \downarrow, \updownarrow\}$. For convenience, we also allow a rule to contain zero or more instances of $(u)_\beta$. For example, if $u = \lambda$, i.e. the *empty* multiset of objects, this rule can be abbreviated as $s\ x \rightarrow_\alpha s'\ x'$.

A cell *evolves* by applying one or more rules, which can change its content and state and can send objects to its neighbors. For a cell $\sigma_i = (Q_i, s_i, w_i, R_i)$, a rule $s\ x \rightarrow_\alpha s'\ x'\ (u)_\beta \mid z \in R_i$ is *applicable*, if $s = s_i$, $x \subseteq w_i$, $z \subseteq w_i$, $\delta(i) \neq \emptyset$ for $\beta = \downarrow$, $\delta^{-1}(i) \neq \emptyset$ for $\beta = \uparrow$ and $\delta(i) \cup \delta^{-1}(i) \neq \emptyset$ for $\beta = \updownarrow$.

The application of a rule transforms the current state s to the *target state* s' transforms multiset x to x' and sends multiset u as specified by the transfer operator β (as further described below). Note that, multisets x' and u will not be visible to other applicable rules in this same step, but they will be visible after all the applicable rules have been applied.

The rules are applied in the *weak priority* order [14], i.e. (1) higher priority applicable rules are applied before lower priority applicable rules, and (2) a lower priority applicable rule is applied only if it indicates the same target state as the previously applied rules.

The rewriting operator $\alpha = \texttt{max}$ indicates that an applicable rewriting rule of R_i is applied as many times as possible. The rewriting operator $\alpha = \texttt{min}$ indicates that an applicable rewriting rule of R_i is applied once. If the right-hand side of a rule contains $(u)_\beta$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$, then for each application of this rule, a copy of multiset u is replicated and sent to each cell $\sigma_j \in \delta^{-1}(i)$ if $\beta = \uparrow$, $\sigma_j \in \delta(i)$ if $\beta = \downarrow$ and $\sigma_j \in \delta(i) \cup \delta^{-1}(i)$ if $\beta = \updownarrow$.

All applicable rules are applied in one *step*. An *execution* of a P system is a sequence of steps, that starts from the initial configuration. An execution *halts* if no further rules are applicable for all cells.

Problem 2. We formulate the FSSP to P systems as follows:
Input: An integer $n \geq 2$ and an integer g, $1 \leq g \leq n$.
Output: A class $\mathcal{C}$ of P systems that satisfies the following two conditions for any

weakly connected digraph (X, A), isomorphic to the structure of a member of $\mathcal{C}$ with $n = |X|$ cells.

1. Cell σ_g is the only cell with an applicable rule (i.e. σ_g can evolve) from its initial configuration.
2. There exists state $s_f \in Q_i$, for all $\sigma_i \in K$, such that during the last step of the system's execution, all cells enter state s_f, simultaneously and for the first time.

We want to find a general-purpose solution to the FSSP that synchronizes in the fewest number of steps, as a function of some of the natural structural properties of a weakly-connected digraph (X, A), such as the eccentricity of node $g \in X$ in the communication graph defined by A.

3 Deterministic FSSP solution for rooted trees

We first solve Problem 2 for the subclass of weakly-connected digraphs (X, A), where the underlying graph of (X, A) is a tree. This section is organized as follow. In Section 3.1, we present the P system for solving the FSSP for trees rooted at the general. In order to help the comprehension of our FSSP algorithm, we provide a trace of the FSSP algorithm in Table 1. Phase I of our FSSP algorithm is described in Section 3.2, which finds the *middle* cell (i.e. a center of a tree, closest to the root) and determines the height of the middle cell. Phase II of our FSSP algorithm is described in Section 3.3, which broadcasts the "command" that prompts all cells to enter the firing state. Finally, in Section 3.4, we present some empirical results that show improvements of our algorithm over the previously best-known FSSP algorithms for tree-based P systems [1, 5].

3.1 P systems for solving the FSSP for rooted trees

Given a tree (X, A) and $g \in X$, our FSSP algorithm is implemented using the P system $\Pi = (O, K, \delta)$ of order $n = |X|$, where:

1. $O = \{a, b, c, e, h, o, v, w\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.
3. δ is a rooted tree, with an underlying graph isomorphic to (X, A), where the general $\sigma_g \in K$ (the root of δ) corresponds to $g \in X$.

All cells have the same set of states, the same set of rules and start at the same initial *quiescent* state s_0, but with different initial contents. The first output condition of Problem 2 will be satisfied by our chosen set of rules.

For each cell $\sigma_i \in K$, its initial configuration is $\sigma_i = (Q, s_0, w_{i0}, R)$ and its final configuration at the end of the execution is $\sigma_i = (Q, s_6, \emptyset, R)$, where:

- $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, where s_0 is the initial quiescent state and s_6 is the *firing* state.

- $w_{i0} = \begin{cases} \{o\} \text{ if } \sigma_i = \sigma_g, \\ \emptyset \text{ if } \sigma_i \neq \sigma_g. \end{cases}$
- R is defined by the following rulesets.
  Rules used in Phase I: all the rules in states s_0, s_1, s_2, s_3 and rule 4.6 in state s_4.
  Rules used in Phase II: all the rules in states s_4 and s_5, except rule 4.6.

0. Rules in state s_0:
   1. $s_0\ o \rightarrow_{\texttt{max}} s_1\ ahou\ (b)_{\downarrow}$
   2. $s_0\ b \rightarrow_{\texttt{max}} s_1\ ah\ (e)_{\uparrow}\ (b)_{\downarrow}$
   3. $s_0\ b \rightarrow_{\texttt{max}} s_4\ a\ (ce)_{\uparrow}$
1. Rules in state s_1:
   1. $s_1\ a \rightarrow_{\texttt{max}} s_2\ ah$
2. Rules in state s_2:
   1. $s_2\ aaa \rightarrow_{\texttt{max}} s_4\ a$
   2. $s_2\ aa \rightarrow_{\texttt{max}} s_3\ a$
   3. $s_2\ ceu \rightarrow_{\texttt{max}} s_2$
   4. $s_2\ ce \rightarrow_{\texttt{max}} s_2$
   5. $s_2\ aee \rightarrow_{\texttt{max}} s_2\ aeeh$
   6. $s_2\ aeooo \rightarrow_{\texttt{max}} s_2\ aa\ (o)_{\downarrow}$
   7. $s_2\ aeou \rightarrow_{\texttt{max}} s_2\ aa\ (o)_{\downarrow}$
   8. $s_2\ aeo \rightarrow_{\texttt{max}} s_2\ aehoo$
   9. $s_2\ ao \rightarrow_{\texttt{max}} s_2\ aaa$
   10. $s_2\ ae \rightarrow_{\texttt{max}} s_2\ aeh$
   11. $s_2\ a \rightarrow_{\texttt{max}} s_2\ aa\ (c)_{\uparrow}$
   12. $s_2\ u \rightarrow_{\texttt{max}} s_2$
3. Rules in state s_3:
   1. $s_3\ a \rightarrow_{\texttt{max}} s_4\ a$
   2. $s_3\ h \rightarrow_{\texttt{max}} s_4$
4. Rules in state s_4:
   1. $s_4\ hh \rightarrow_{\texttt{max}} s_5\ w\ (v)_{\updownarrow}$
   2. $s_4\ avv \rightarrow_{\texttt{max}} s_5\ aw\ (v)_{\updownarrow}$
   3. $s_4\ avv \rightarrow_{\texttt{max}} s_5\ aw$
   4. $s_4\ av \rightarrow_{\texttt{max}} s_6$
   5. $s_4\ v \rightarrow_{\texttt{max}} s_5\ w\ (v)_{\updownarrow}$
   6. $s_4\ o \rightarrow_{\texttt{max}} s_4$
5. Rules in state s_5:
   1. $s_5\ aww \rightarrow_{\texttt{max}} s_5\ aw$
   2. $s_5\ aw \rightarrow_{\texttt{max}} s_6$
   3. $s_5\ v \rightarrow_{\texttt{max}} s_6$
   4. $s_5\ o \rightarrow_{\texttt{max}} s_6$

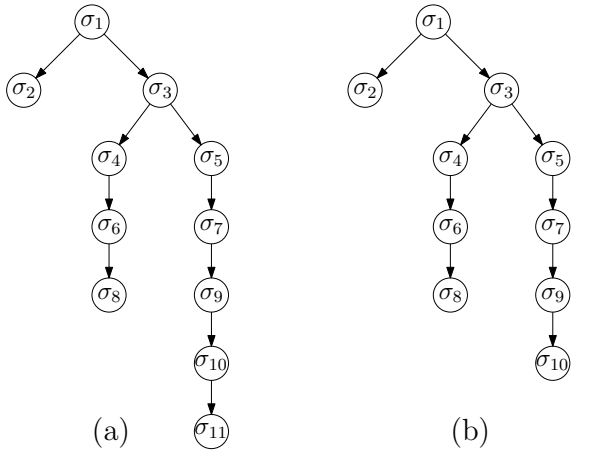


Fig. 1. (a) a tree with the center σ_5; (b) a tree with two centers σ_3 and σ_5, σ_3 being the middle cell.

Table 1. The traces of the FSSP algorithm on a P system with the membrane structure defined by the tree shown in Figure 1 (a), where the general is σ_1 and the middle cell is σ_5. The step in which the Phase I ends (or the Phase II begins) is indicated by the shaded table cells.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}
0	$s_0\ o$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ ahou$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_2\ ace^2h^2ou$	$s_4\ a$	$s_1\ ah$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0
3	$s_2\ aeh^3o^2$	$s_4\ a$	$s_2\ ae^2h^2$	$s_1\ ah$	$s_1\ ah$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0
4	$s_2\ aeh^4o^3$	$s_4\ a$	$s_2\ ae^2h^3$	$s_2\ aeh^2$	$s_2\ aeh^2$	$s_1\ ah$	$s_1\ ah$	$s_0\ b$	$s_0\ b$	s_0	s_0
5	$s_2\ a^2h^4$	$s_4\ ao$	$s_2\ ae^2h^4o$	$s_2\ aeh^3$	$s_2\ aeh^3$	$s_2\ aceh^2$	$s_2\ aeh^2$	$s_4\ a$	$s_1\ ah$	$s_0\ b$	s_0
6	$s_3\ ah^4$	$s_4\ a$	$s_2\ ae^2h^5o$	$s_2\ aceh^4$	$s_2\ aeh^4$	$s_2\ a^2h^2$	$s_2\ aeh^3$	$s_4\ a$	$s_2\ aeh^2$	$s_1\ ah$	$s_0\ b$
7	$s_4\ a$	$s_4\ a$	$s_2\ ace^2h^6o$	$s_2\ a^2h^4$	$s_2\ aeh^5$	$s_3\ ah^2$	$s_2\ aeh^4$	$s_4\ a$	$s_2\ aeh^3$	$s_2\ aceh^2$	$s_4\ a$
8	$s_4\ a$	$s_4\ a$	$s_2\ aeh^7o^2$	$s_3\ ah^4$	$s_2\ aeh^6$	$s_4\ a$	$s_2\ aeh^5$	$s_4\ a$	$s_2\ aceh^4$	$s_2\ a^2h^2$	$s_4\ a$
9	$s_4\ a$	$s_4\ a$	$s_2\ aeh^8o^3$	$s_4\ a$	$s_2\ aeh^7$	$s_4\ a$	$s_2\ aceh^6$	$s_4\ a$	$s_2\ a^2h^4$	$s_3\ ah^2$	$s_4\ a$
10	$s_4\ a$	$s_4\ a$	$s_2\ a^2h^8$	$s_4\ ao$	$s_2\ aceh^8o$	$s_4\ a$	$s_2\ a^2h^6$	$s_4\ a$	$s_3\ ah^4$	$s_4\ a$	$s_4\ a$
11	$s_4\ a$	$s_4\ a$	$s_3\ ah^8$	$s_4\ a$	$s_2\ a^3h^8$	$s_4\ a$	$s_3\ ah^6$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$
12	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ ah^8$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$
13	$s_4\ a$	$s_4\ a$	$s_4\ av^4$	$s_4\ a$	$s_5\ aw^4$	$s_4\ a$	$s_4\ av^4$	$s_4\ a$	$s_4\ a$	$s_4\ a$	$s_4\ a$
14	$s_4\ av^3$	$s_4\ a$	$s_5\ aw^3$	$s_4\ av^3$	$s_5\ av^6w^3$	$s_4\ a$	$s_5\ aw^3$	$s_4\ a$	$s_4\ av^3$	$s_4\ a$	$s_4\ a$
15	$s_5\ aw^2$	$s_4\ av^2$	$s_5\ av^4w^2$	$s_5\ aw^2$	$s_5\ av^6w^2$	$s_4\ av^2$	$s_5\ av^2w^2$	$s_4\ a$	$s_5\ aw^2$	$s_4\ av^2$	$s_4\ a$
16	$s_5\ avw$	$s_5\ aw$	$s_5\ av^4w$	$s_5\ avw$	$s_5\ av^6w$	$s_5\ aw$	$s_5\ av^2w$	$s_4\ av$	$s_5\ avw$	$s_5\ aw$	$s_4\ av$
17	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6	s_6

3.2 Phase I: Find the middle cell of rooted trees

In this phase, a breadth-first search (BFS) is performed from the root, which propagates symbol b from the root to all other cells. When the symbol b from the BFS reaches a leaf cell, symbol c is *reflected* back up the tree. Starting from the root, the search for the middle cell is performed as described below, where symbol o represents the current *search pivot*. Note that symbol o's propagation speed is 1/3 of the propagation speed of symbols b and c; intuitively, this ensures that o and c meet in the middle cell.

We provide a visual description of the propagations of symbols b, c and o in Figure 4 (for a tree with one center) and Figure 3 (for a tree with two centers).

Details of Phase I

Objective: The objective of Phase I is to find the middle cell, σ_m, and its height, $\texttt{height}_g(m)$.

Precondition: Phase I starts with the initial configuration of P system Π, described in Section 3.1.

Postcondition: Phase I ends when σ_m enters state s_4. At the end of Phase I, the configuration of cell $\sigma_i \in K$ is (Q, s_4, w_i, R), where $|w_i|_a = 1$; $|w_i|_h = 2{\cdot}\texttt{height}_g(i)$, if $\sigma_i = \sigma_m$.

Description: In Phase I, each cell starts in state s_0, transits through states s_1, s_2, s_3, and ends in state s_4; a cell in state s_4 will ignore any symbol o that it may receive.

The behaviors of cells in this phase are described below.

- **Propagation of symbol b:** The root cell sends symbol b to all its children (Rule 0.1). An internal cell forwards the received symbol b to all its children (Rule 0.2) After applying Rule 0.1 or 0.2, each of these non-leaf cells produces a copy of symbol h in each step, until it receives symbol c from all its children (Rules 1.1 and 2.10).
- **Propagation of symbol c:** If a leaf cell receives symbol b, then it sends symbol c to its parent (Rule 0.3) and enters state s_4 (the end state of Phase I). If a non-leaf cell receives symbol c from all its children, then it sends symbol c to its parent (Rules 2.4 and 2.11), consumes all copies of symbol h and enters state s_4 (Rule 3.2).
- Note, when a cell applies Rule 0.2 or 0.3, it sends one copy of symbol e up to its parent. A copy of symbol e is consumed with a copy of symbol c by Rule 2.4. Hence, $|w_i|_e = k$ indicates the number of σ_i's children that have not sent symbol c to σ_i.
- **Propagation of symbol o:** The root cell initially contains the symbol o. We denote σ_j as the current cell that contains symbol o and has not entered state s_4.

  Assume, at step t, σ_j received symbol c from all but one subtree rooted at σ_v. Starting from step $t+1$, σ_j produces a copy of symbol o in each step, until it

receives symbol c from σ_v (Rule 2.8), That is, $|w_j|_o - 1$ indicates the number of steps since σ_j received symbol c from all of its children except σ_v.
If σ_j receives symbol c from σ_v by step $t+2$, i.e. $|w_j|_o \leq 3$, then σ_j is the middle cell; σ_j keeps all copies of symbol h and enters state s_4 (Rule 2.1). Otherwise, σ_j sends a copy of symbol o to σ_v at step $t+3$ (Rule 2.6 or 2.7); in the subsequent steps, σ_j consumes all copies of symbol h and enters state s_4 (Rules 2.2 and 3.2). Note, using current setup, σ_j cannot send a symbol to a specific child; σ_j has to send a copy of symbol o to all its children. However, all σ_j's children, except σ_v, would have entered state s_4.

Proposition 1 indicates the step in which σ_m receives symbol c from all its children and Proposition 2 indicates the number of steps needed to propagate symbol o from σ_g to σ_m.

Proposition 1. *Cell σ_m receives the symbol c from all its children by step* $\texttt{height}_g(g) + \texttt{height}_g(m)$.

Proof. Cell σ_m is at distance $\texttt{height}_g(g) - \texttt{height}_g(m)$ from σ_g, hence σ_m receives symbol b in step $\texttt{height}_g(g) - \texttt{height}_g(m)$. In the subtree rooted at σ_m, the propagations of the symbol b from σ_m to its farthest leaf and the symbol c reflected from the leaf to σ_m take $2 \cdot \texttt{height}_g(m)$ steps. Thus, σ_m receives symbol c from all its children by step $\texttt{height}_g(g) - \texttt{height}_g(m) + 2 \cdot \texttt{height}_g(m) = \texttt{height}_g(g) + \texttt{height}_g(m)$. □

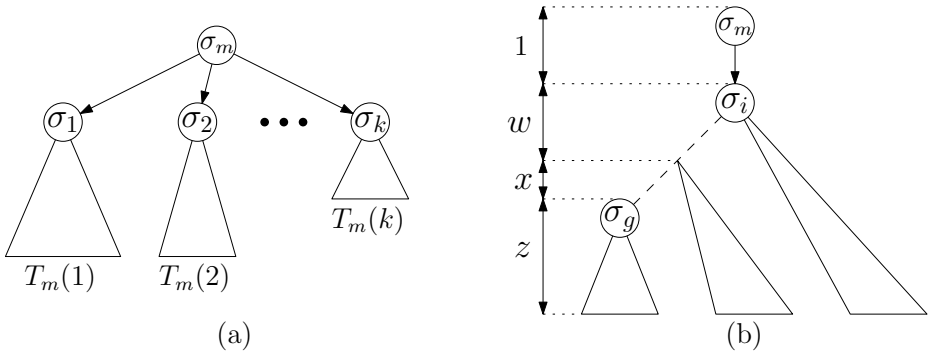


Fig. 2. (a) k subtrees of σ_m, $T_m(1), T_m(2), \ldots, T_m(k)$. (b) The structure of subtree $T_m(j)$, which contains σ_g.

Proposition 2. *The propagation of the symbol o from σ_g to σ_m takes at most* $\texttt{height}_g(g) + \texttt{height}_g(m)$ *steps.*

Proof. For a given tree T_g, rooted at σ_g, we construct a tree T_m, which re-roots T_g at σ_m. Recall, $T_m(i)$ denotes a subtree rooted at σ_i in T_m. Assume that σ_m has $k \geq 2$ subtrees, $T_m(1), T_m(2), \ldots, T_m(k)$, such that $\texttt{height}_m(1) \geq \texttt{height}_m(2) \geq$

$\cdots \geq \texttt{height}_m(k)$ and $\texttt{height}_m(1) - \texttt{height}_m(2) \leq 1$. Figure 2 (a) illustrates the subtrees of σ_m.

Assume $T_m(i)$ is a subtree of σ_m, which contains σ_g. In $T_m(i)$, let z be the height of σ_g and $x + w \geq 0$ be the distance between σ_g and σ_i. Figure 2 (b) illustrates the z, x and w in $T_m(i)$.

In $T_m(i)$, let p be a path from σ_i to its farthest leaf and t be the number of steps needed to propagate symbol o from σ_g to σ_m. Note, $\texttt{height}_m(m) = \texttt{height}_g(m)$ and $x + w + 1 = \texttt{height}_g(g) - \texttt{height}_g(m)$.

We have three cases to consider to prove Proposition 2.

1. $\texttt{height}_m(i) = \texttt{height}_m(m) - 1$.
   - If σ_g is a part of path p, then $z + x + w + 1 = \texttt{height}_m(m)$, hence

$$\begin{aligned} 2z + 3(x + w + 1) &= 2(z + x + w + 1) + (x + w + 1) \\ &= 2 \cdot \texttt{height}_m(m) + (\texttt{height}_g(g) - \texttt{height}_g(m)) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) \end{aligned}$$

   - If σ_g is not a part of p, then $(v - w) + w + 1 = v + 1 = \texttt{height}_m(m)$, hence

$$\begin{aligned} x + 2(v - w) + 3(w + 1) &= 2(v + 1) + (x + w + 1) \\ &= 2 \cdot \texttt{height}_m(m) + (\texttt{height}_g(g) - \texttt{height}_g(m)) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) \end{aligned}$$

   Cell σ_m receives symbol o in step $\texttt{height}_g(g) + \texttt{height}_g(m)$.
2. $\texttt{height}_m(i) = \texttt{height}_m(m) - 2$.
   - If σ_g is a part of p, then $z + x + w + 1 = \texttt{height}_m(m) - 1$, hence

$$\begin{aligned} 2z + 3(x + w + 1) &= 2(z + x + w + 1) + (x + w + 1) \\ &= 2 \cdot \texttt{height}_m(m) - 2 + \texttt{height}_g(g) - \texttt{height}_g(m) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) - 2 \end{aligned}$$

   - If σ_g is not a part of p, then $(v - w) + w + 1 = v + 1 = \texttt{height}_m(m) - 1$, hence

$$\begin{aligned} x + 2(v - w) + 3(w + 1) &= 2(v + 1) + (x + w + 1) \\ &= 2 \cdot \texttt{height}_m(m) - 2 + \texttt{height}_g(g) - \texttt{height}_g(m) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) - 2 \end{aligned}$$

   Note, symbol o remains in σ_m for at least two steps. Thus, symbol o, arrived in σ_m at step $\texttt{height}_g(g) + \texttt{height}_g(m) - 2$, will remain in σ_m until step $\texttt{height}_g(g) + \texttt{height}_g(m)$.
3. $\texttt{height}_m(i) = \texttt{height}_m(m) - j$, $j \geq 3$.

- If σ_g is a part of p, then $z + x + w + 1 = \texttt{height}(m) - j + 1$, hence

$$\begin{aligned} 2z + 3(x + w + 1) &= 2(z + x + w + 1) + (x + w + 1) \\ &= 2 \cdot \texttt{height}_g(m) - 2j + 2 + \texttt{height}_g(g) - \texttt{height}_g(m) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) - 2j + 2 \end{aligned}$$

- If σ_g is not a part of p, then $(v - w) + w + 1 = v + 1 = \texttt{height}_g(m) - j + 1$, hence

$$\begin{aligned} x + 2(v - w) + 3(w + 1) &= 2(v + 1) + (x + w + 1) \\ &= 2 \cdot \texttt{height}_g(m) - 2j + 2 + \texttt{height}_g(g) - \texttt{height}_g(m) \\ &= \texttt{height}_g(g) + \texttt{height}_g(m) - 2j + 2 \end{aligned}$$

In T_m, σ_m has two subtrees, $T_m(1)$ and $T_m(2)$, such that $\texttt{height}_m(1) = \texttt{height}_m(m) - 1$ and $\texttt{height}_m(1) - \texttt{height}_m(2) \leq 1$.
The symbol o arrived in σ_m at step $\texttt{height}_g(g) + \texttt{height}_g(m) - 2j + 2$, $j \geq 3$, will remain in σ_m until step $\texttt{height}_g(g) + \texttt{height}_g(m)$.

□

Proposition 3. *Phase I takes* $\texttt{height}_g(g) + \texttt{height}_g(m) + 2$ *steps.*

Proof. From Propositions 1 and 2, symbols o and c meets in σ_m at step $\texttt{height}_g(g) + \texttt{height}_g(m)$. Cell σ_m enters state s_4 by applying Rule 2.9 and 2.1, which takes two steps. Thus, Phase I takes $\texttt{height}_g(g) + \texttt{height}_g(m) + 2$ steps. □

3.3 Phase II: Determine the step to enter the firing state

Phase II begins immediately after Phase I. In Phase II, the middle cell broadcasts the "firing" order, which prompts receiving cells to enter the firing state. In general, the middle cell does not have direct communication channels to all cells. Thus, the firing order has to be relayed through intermediate cells, which results in some cells receiving the order before other cells. To ensure that all cells enter the firing state simultaneously, each cell needs to determine the number of steps it needs to wait, until all other cells receive the order.

The firing order is paired with a *counter*, which is initially set to the eccentricity of the middle cell. Propagating an order from one cell to another decrements its current counter by one. The current counter of the received order equals the number of remaining steps before all other cells receive the order. Hence, each cell waits according to the current counter, before it enters the firing state. Figure 5 illustrates the propagation of the firing order.

Details of Phase II

Objective: The objective of Phase II is to determine the step to enter the firing state, such that during the last step of Phase II, i.e. the system's execution, all cells enter the firing state, simultaneously and for the first time.

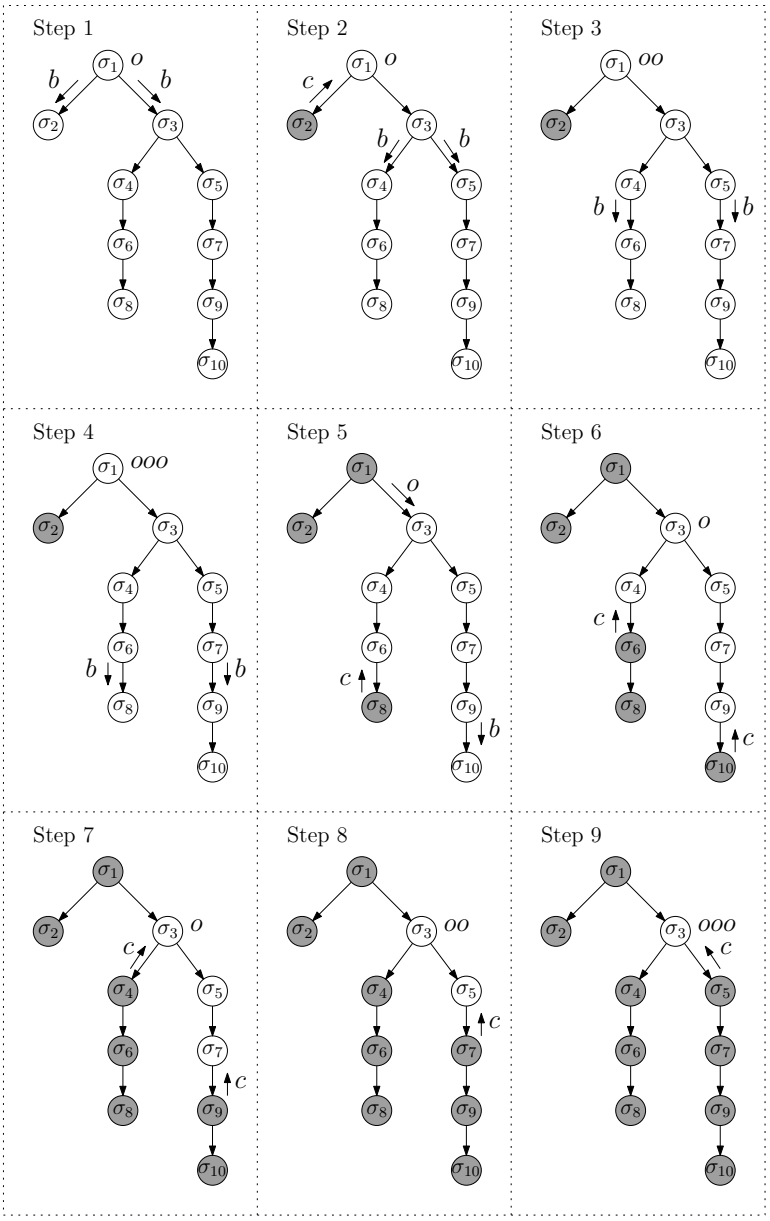


Fig. 3. Propagations of symbols b, c and o, in a tree with two centers. The symbols c and o meet at the middle cell σ_3. Cells that have sent symbol c or o are shaded. The propagation of symbol o to a shaded cell is omitted. In cell σ_j, $j \in \{1,3\}$, $|w_j|_o - 1$ represents the number of steps since σ_j received symbol c from all of its children but one.

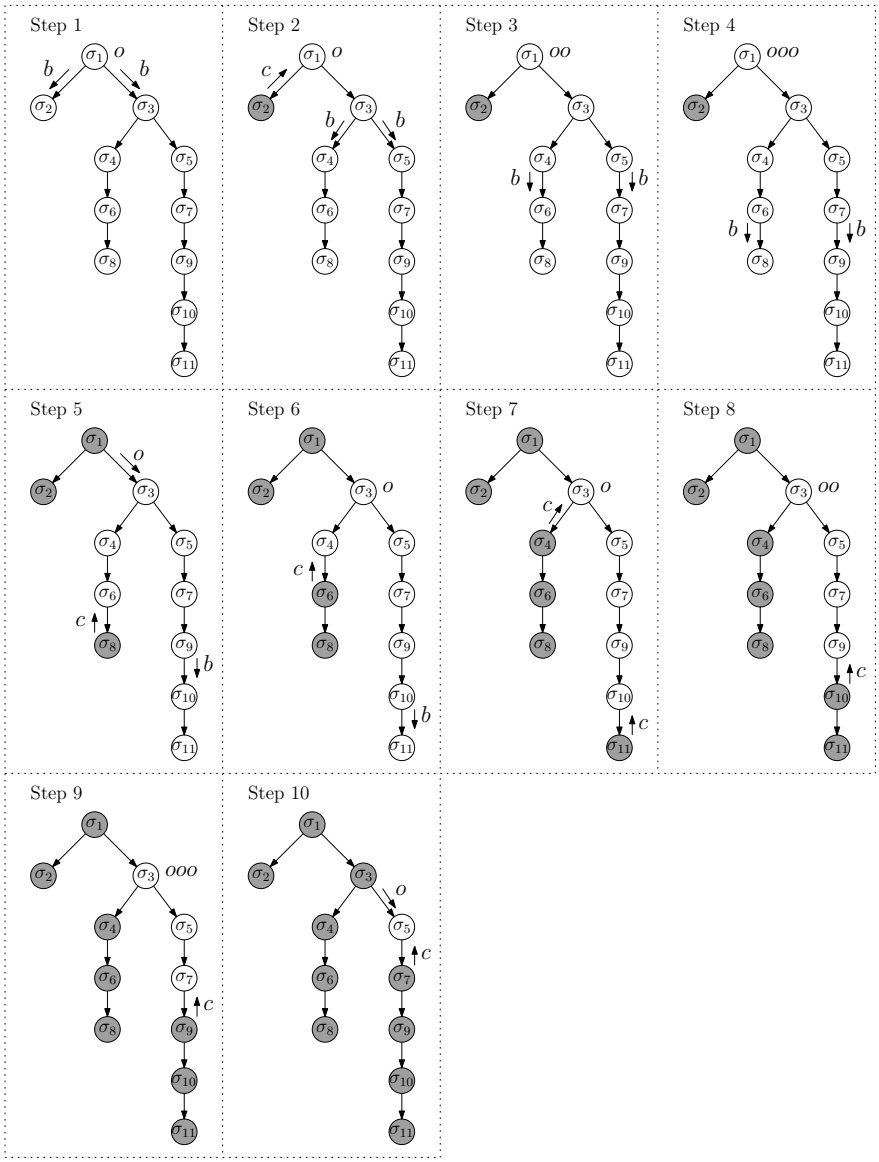


Fig. 4. Propagations of symbols b, c and o, in a tree with one center. The symbols c and o meet at the middle cell σ_5. Cells that have sent symbol c or o are shaded. The propagation of symbol o to a shaded cell is omitted. In cell σ_j, $j \in \{1,3\}$, $|w_j|_o - 1$ represents the number of steps since σ_j received symbol c from all of its children but one.

Precondition: Phase II starts with the postcondition of Phase I, described in Section 3.2.

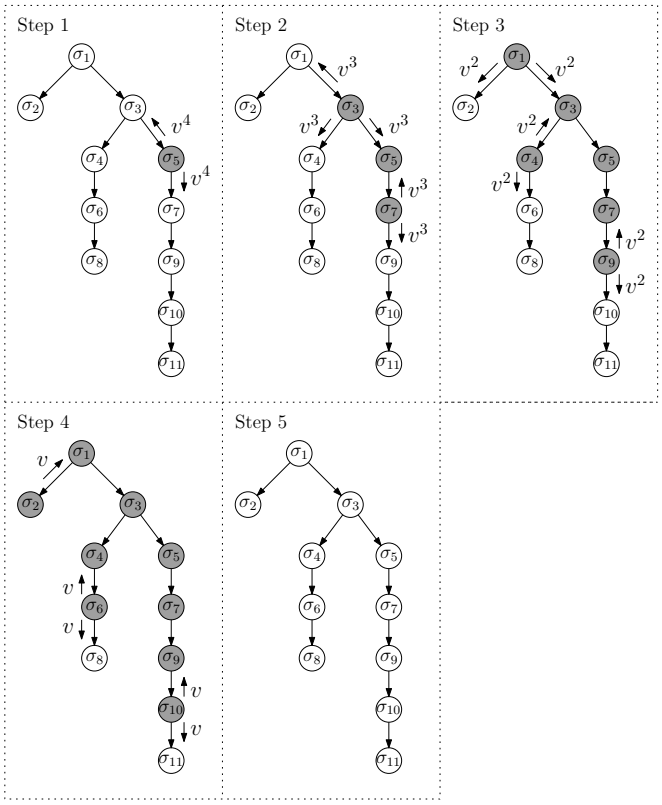


Fig. 5. Propagations of the firing order from the middle cell, σ_5, where the counter is represented by the multiplicity of symbol v. Cells that have propagated the order are shaded.

Postcondition: Phase II ends when all cells enter the firing state s_6. At the end of Phase II, the configuration of cell $\sigma_i \in K$ is $(Q, s_6, \emptyset, R)$.

Description: The behaviors of the middle cell σ_m and a non-middle cell, $\sigma_i \neq \sigma_m$, in this phase are as follow. We also indicate which rules accomplish the described behaviors.

- We first describe the behavior of σ_m. For every two copies of symbol h, σ_m produces one copy of symbol w and sends one copy of symbol v to all its neighbors (Rules 4.1 and 4.2). In the next sequence of steps, σ_m consumes one copy of symbol w (Rule 5.1). If σ_m consumes all copies of symbol w, then σ_m enters the firing state (Rule 5.2).
- Next, we describe the behavior of $\sigma_i \neq \sigma_m$. Let $k_i \geq 1$ denote the multiplicity of symbol v that σ_i receives for the first time. If $k_i = 1$, then σ_i enters the firing state (Rule 4.6). If $k_i \geq 2$, then σ_i consumes k_i copies of symbol v, produces $k_i - 1$ copies of symbol w and sends $k_i - 1$ copies of symbol v to all its neighbors

(Rules 4.3, 4.4, 4.7 and 4.8); in each subsequent step, σ_i consumes one copy of symbol w (Rule 5.1) and σ_i enters the firing state (Rule 5.2), after all copies of symbol w is consumed.

Proposition 4. *Cell σ_m produces $\texttt{height}_g(m)$ copies of symbol w and sends $\texttt{height}_g(m)$ copies of symbol v to all is neighbors.*

Proof. At the beginning of Phase II, σ_m contains $2 \cdot \texttt{height}_g(m)$ copies of symbol h. As described earlier, for every two copies of the symbol h that σ_m consumes, σ_m produces one copy of symbol w and sends one copy of symbol v to all its neighbors. □

Proposition 5. *Cell σ_i receives k copies of symbol v at step t and sends $k-1$ copies of symbol v to all its neighbors at step $t+1$, where $k = \texttt{height}_g(m) - \texttt{depth}_m(i) + 1$ and $t = \texttt{height}_g(g) + \texttt{height}_g(m) + \texttt{depth}_m(i) + 2$.*

Proof. Proof by induction on $\texttt{depth}_m(i) \geq 1$. First, σ_m sends $\texttt{height}_g(m)$ copies of symbol v to all its neighbors. Thus, each cell σ_i, at distance 1 from σ_m, receives $\texttt{height}_g(m)$ copies of symbol v. By Rule $4.3, 4.4, 4.7, 4.8$, σ_i consumes $\texttt{height}_g(m)$ copies of symbol v, produces $\texttt{height}_g(m) - 1$ copies of symbol w and sends $\texttt{height}_g(m) - 1$ copies of symbol v to all its neighbors.

Assume that the induction hypothesis holds for each cell σ_j at distance $\texttt{depth}_m(j)$. Consider cell σ_i, where $\texttt{depth}_m(i) = \texttt{depth}_m(j) + 1$. By the induction hypothesis, cell $\sigma_j \in \texttt{Neighbor}(i)$, sends $\texttt{height}_g(m) - \texttt{depth}_m(j) = \texttt{height}_g(m) - \texttt{depth}_m(i) + 1$ copies of symbol v, such that σ_i receives $\texttt{height}_g(m) - \texttt{depth}_m(i) + 1$ copies of symbol v. By Rule $4.3, 4.4, 4.7, 4.8$, σ_i consumes $\texttt{height}_g(m) - \texttt{depth}_m(i) + 1$ copies of symbol v, produces $\texttt{height}_g(m) - \texttt{depth}_m(i)$ copies of symbol w and sends $\texttt{height}_g(m) - \texttt{depth}_m(i)$ copies of symbol v to all its neighbors. □

Proposition 6. *Phase II takes $\texttt{height}_g(m) + 1$ steps.*

Proof. Each cell σ_i receives $\texttt{height}_g(m) - \texttt{depth}_m(i) + 1$ copies of symbol v at step $\texttt{height}_g(g) + \texttt{height}_g(m) + \texttt{depth}_m(i) + 2$.

Consider σ_j, where $\texttt{depth}_m(j) = \texttt{height}_g(m)$. Cell σ_j receives one copy of symbol v. As described earlier, if a cell receives one copy of symbol v, then it enters the firing state at the next step. Hence, σ_j enters the firing state at step $\texttt{height}_g(g) + 2 \cdot \texttt{height}_g(m) + 3$.

Consider σ_k, where $\texttt{depth}_m(k) < \texttt{height}_g(m)$. Cell σ_k contains $\texttt{height}_g(m) - \texttt{depth}_m(i)$ copies of symbol w at step $\texttt{height}_g(g) + \texttt{height}_g(m) + \texttt{depth}_m(i) + 3$. Since σ_k consumes one copy of symbol w in each step, σ_k will take $\texttt{height}_g(m) - \texttt{depth}_m(i)$ steps to consume all copies of symbol w. Hence, σ_j enters the firing state at step $(\texttt{height}_g(g) + \texttt{height}_g(m) + \texttt{depth}_m(i) + 3) + (\texttt{height}_g(m) - \texttt{depth}_m(i)) = \texttt{height}_g(g) + 2 \cdot \texttt{height}_g(m) + 3$.

Phase I ends at step $\texttt{height}_g(g) + \texttt{height}_g(m) + 2$ and all cells enter the firing state at step $\texttt{height}_g(g) + 2 \cdot \texttt{height}_g(m) + 3$. Thus, Phase II takes $\texttt{height}_g(m) + 1$ steps. □

Theorem 3. *The synchronization time of our FSSP solution, for a P system with underlying structure of a tree, is $\texttt{height}_g(g) + 2 \cdot \texttt{height}_g(m) + 3$.*

Proof. The result is obtained by summing the individual running times of Phases I and II, as given by Propositions 3 and 6: $(\texttt{height}_g(g) + \texttt{height}_g(m) + 2) + (\texttt{height}_g(m) + 1) = \texttt{height}_g(g) + 2 \cdot \texttt{height}_g(m) + 3$. □

3.4 Empirical results

We tested the improvement in running times over the previously best-known FSSP algorithms that synchronize tree-based P systems [1, 5]. We wanted to see how our new running time, that is proportional to $e + 2r$, compares with the earlier value of $3e$, where e is the eccentricity of the general (which is also the height of the tree, rooted at the general) and r is the radius of a tree. We did two tests suites; one for relatively small trees and one for larger trees as shown in Tables 2 and 3, respectively. In both cases, our empirical results show at least 20% reduction in the number of steps needed to synchronize, which we believe is significant.

For the statistics given in Table 2, we generated random (free) trees by starting from a single node and repeatedly add new leaf nodes to the partially generated tree. We then averaged over all possible locations for the general node. The "average gain" is the average difference $3e - (e + 2r)$ and the "average % gain" is improvement as a percentage speedup over $3e$.

Table 2. Statistics for improvement on many random trees of various (smaller) orders.

n	average height	average radius	avgerage 3·height	avgerage height+2·radius	avgerage % gain
100	22.12	14.49	66.36	51.1	23.00
200	31.91	21.35	95.73	74.61	22.06
300	41.13	26.79	123.39	94.71	23.24
400	47.86	31.3	143.58	110.46	23.07
500	51.52	33.77	154.56	119.06	22.97
600	57.16	37.76	171.48	132.68	22.63
700	63.43	42.19	190.29	147.81	22.32
800	68.12	45.37	204.36	158.86	22.26
900	72.46	47.83	217.38	168.12	22.66
1000	79.94	52.21	239.82	184.36	23.13

For the statistics given in Table 3, we generated random labeled trees using the well-known Prüfer correspondence [19] (using the implementation given in Sage [16]). In these sets of trees, the first indexed vertex is randomly placed, unlike the random trees generated in our first test suite. Hence, for this test suite, we did not need to average over all possible general node locations per tree. Due

to the uniform randomness of the labeled tree generator, we assumed the general is placed at the node labeled by 1. Each row in Table 3 is based on 100 random trees of that given order.

We have run both test suites several times and the results are consistent with these two tables. Hence, we are pretty confident in the practical speedup that our new synchronization algorithm provides.

Table 3. Statistics for improvement on random trees of various (larger) orders.

n	diameter	radius	avgerage eccentricity	avgerage gain	avgerage % gain
1000	21	11	16.27	10.54	21.59
2000	32	16	23.45	14.90	21.18
3000	26	13	19.97	13.95	23.27
4000	30	15	22.65	15.30	22.51
5000	35	18	26.51	17.01	21.40
6000	32	16	23.82	15.64	21.89
7000	34	17	25.29	16.58	21.85
8000	34	17	25.01	16.03	21.36
9000	40	20	28.37	16.74	19.67
10000	37	19	27.16	16.32	20.03
10000	38	19	27.36	16.72	20.37
20000	37	19	28.23	18.47	21.80
30000	43	22	31.74	19.49	20.46
40000	43	22	31.55	19.09	20.18
50000	42	21	30.81	19.63	21.23
60000	44	22	32.50	21.00	21.54
70000	48	24	34.55	21.09	20.35
80000	45	23	33.08	20.17	20.32
90000	50	25	36.00	22.01	20.37
100000	47	24	34.15	20.29	19.81

4 FSSP solution for digraphs

The key idea of FSSP solution for digraphs is as follows. For a given digraph, perform a BFS from the general on the communication graph and construct a *virtual* spanning tree, implemented via pointer symbols, not by changing existing

arcs. If a node finds multiple parents in the BFS, then one of the parents is chosen as its spanning tree parent. In Figure 6, (a) illustrates a digraph G, (b) illustrates the underlying graph of G and (c) illustrates a spanning tree of the underlying graph of G, rooted at σ_1.

Using the spanning tree constructed from the BFS, the FSSP algorithm described in Section 3, is applied to achieve the synchronization.

We present the details of P system for solving the FSSP (Problem 2) for digraphs in Section 4.1. A trace of the FSSP algorithm for digraphs is given in Table 4. The details Phases I and II of this FSSP algorithm are described in Sections 4.2 and 4.3, respectively. Finally, in Section 4.4, we present some empirical results that illustrates expected improvements of our new algorithm over our previous FSSP algorithm for digraphs [5].

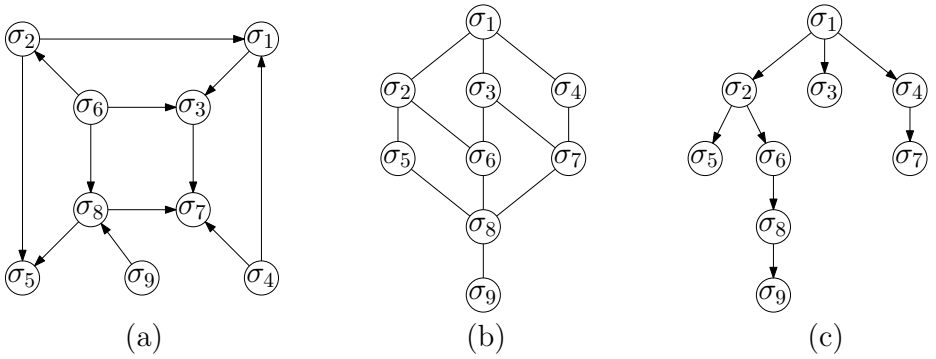


Fig. 6. (a) A digraph G. (b) The underlying graph of G. (c) A spanning tree of the underlying graph of G, rooted at σ_1.

4.1 P systems for solving the FSSP for digraphs

Given a digraph (X, A) and $g \in X$, our FSSP algorithm is implemented using the P system $\Pi' = (O, K, \delta)$ of order $n = |X|$, where:

1. $O = \{a, h, o, v, w, x, z\} \cup \{\iota_k, b_k, c_k, e_k, p_k \mid 1 \leq k \leq n\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.
3. δ is a digraph, isomorphic to (X, A), where the general $\sigma_g \in K$ corresponds to $g \in X$.

All cells have the same set of states and start at the same initial quiescent state s_0, but with different initial contents and set of rules. The first output condition of Problem 2 will be satisfied by our chosen set of rules.

In this FSSP solution, we extend the basic P module framework, described Section 2. Specifically, we assume that each cell $\sigma_i \in K$ has a unique *cell ID* symbol ι_i, which will be used as an *immutable promoter* and we allow rules with a simple form of *complex symbols*.

To explain these additional features, consider rules 3.10 and 3.11 from the ruleset R, listed below. In this ruleset, symbols i and j are *free variables* (which in our case happen to match cell IDs). Symbols e_i and e_j are complex symbols. Rule 3.11 deletes all existing e_j symbols, regardless of the actual values matched by the free variable j. However, the preceding rule 3.10 fires only for symbols e_i, with indices i matching the local cell ID, as required by the right-hand side promoter ι_i. Together, rules 3.10 and 3.11, applied in a weak priority scheme, keep all symbols e_i, with indices i matching the local cell ID, and delete all other symbols e_j.

For each cell $\sigma_i \in K$, its initial configuration is $\sigma_i = (Q, s_0, w_{i0}, R)$ and its final configuration at the end of the execution is $\sigma_i = (Q, s_7, \{\iota_i\}, R)$, where:

- $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, where s_0 is the initial quiescent state and s_7 is the firing state.
- $w_{i0} = \begin{cases} \{\iota_g o\} \text{ if } \sigma_i = \sigma_g, \\ \{\iota_i\} \text{ if } \sigma_i \neq \sigma_g. \end{cases}$
- R is defined by the following rulesets.

  Rules used in Phase I: all the rules in states s_0, s_1, s_2, s_3, s_4 and rules 5.5 and 5.6 in state s_5.

  Rules used in Phase II: all the rules in states s_5 and s_6, except rules 5.5 and 5.6.

0. Rules for cells in state s_0:
   1. $s_0\ o \to_{\min} s_1\ ao\ (xb_i)_{\updownarrow} \mid \iota_i$
   2. $s_0\ x \to_{\min} s_1\ a\ (xb_i)_{\updownarrow} \mid \iota_i$
   3. $s_0\ b_j \to_{\max} s_1\ p_j$
1. Rules for cells in state s_1:
   1. $s_1\ ap_j \to_{\max} s_2\ ap_j\ (e_j)_{\updownarrow}$
   2. $s_1\ a \to_{\max} s_2\ a$
   3. $s_1\ p_j \to_{\max} s_2$
2. Rules for cells in state s_2:
   1. $s_2\ a \to_{\max} s_3\ a$
   2. $s_2\ b_j \to_{\max} s_3$
   3. $s_2\ x \to_{\max} s_3$
3. Rules for cells in state s_3:
   1. $s_3\ aaa \to_{\max} s_5\ a$
   2. $s_3\ aa \to_{\max} s_4\ a$
   3. $s_3\ c_i e_i \to_{\max} s_3 \mid \iota_i$
   4. $s_3\ aoooe_i \to_{\max} s_3\ aa\ (o)_{\updownarrow} \mid \iota_i$
   5. $s_3\ aoe_i e_i \to_{\max} s_3\ ahoe_i e_i \mid \iota_i$
   6. $s_3\ aoe_i \to_{\max} s_3\ ahooe_i \mid \iota_i$
   7. $s_3\ ao \to_{\max} s_3\ aaa$
   8. $s_3\ ae_i \to_{\max} s_3\ ae_i h \mid \iota_i$
   9. $s_3\ ap_j \to_{\max} s_3\ aa\ (c_j)_{\updownarrow}$
   10. $s_3\ e_i \to_{\max} s_3\ e_i \mid \iota_i$
   11. $s_3\ e_j \to_{\max} s_3$
   12. $s_3\ p_j \to_{\max} s_4$
   13. $s_3\ p_j \to_{\max} s_5$
4. Rules for cells in state s_4:
   1. $s_4\ a \to_{\max} s_5$
   2. $s_4\ h \to_{\max} s_5$
   3. $s_4\ c_j \to_{\max} s_5$
5. Rules for cells in state s_5:
   1. $s_5\ a \to_{\max} s_6\ a\ (z)_{\updownarrow}$
   2. $s_5\ hh \to_{\max} s_6\ w\ (v)_{\updownarrow}$
   3. $s_5\ zv \to_{\max} s_6\ a\ (z)_{\updownarrow}$
   4. $s_5\ v \to_{\max} s_6\ w\ (v)_{\updownarrow}$
   5. $s_5\ o \to_{\max} s_5$
   6. $s_5\ c_j \to_{\max} s_5$
6. Rules for cells in state s_6:
   1. $s_6\ aw \to_{\max} s_6\ a$
   2. $s_6\ a \to_{\max} s_7$
   3. $s_6\ z \to_{\max} s_7$
   4. $s_6\ v \to_{\max} s_7$

Table 4. The traces of the FSSP algorithm on the digraph of Figure 6 (a), where the general is σ_1 and the middle cell is σ_2. The step in which the Phase I ends (or the Phase II begins) is indicated by the shaded table cells.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	$s_0\ \iota_1 o$	$s_0\ \iota_2$	$s_0\ \iota_3$	$s_0\ \iota_4$	$s_0\ \iota_5$	$s_0\ \iota_6$	$s_0\ \iota_7$	$s_0\ \iota_8$	$s_0\ \iota_9$
1	$s_1\ \iota_1 ao$	$s_0\ \iota_2 b_1 x$	$s_0\ \iota_3 b_1 x$	$s_0\ \iota_4 b_1 x$	$s_0\ \iota_5$	$s_0\ \iota_6$	$s_0\ \iota_7$	$s_0\ \iota_8$	$s_0\ \iota_9$
2	$s_2\ \iota_1 ab_2 b_3 b_4 o x^3$	$s_1\ \iota_2 ap_1$	$s_1\ \iota_3 ap_1$	$s_1\ \iota_4 ap_1$	$s_0\ \iota_5 b_2 x$	$s_0\ \iota_6 b_2 b_3 x^2$	$s_0\ \iota_7 b_3 b_4 x^2$	$s_0\ \iota_8$	$s_0\ \iota_9$
3	$s_3\ \iota_1 ae_1^3 o$	$s_2\ \iota_2 ab_5 b_6 p_1 x^2$	$s_2\ \iota_3 ab_6 b_7 p_1 x^2$	$s_2\ \iota_4 ab_7 p_1 x$	$s_1\ \iota_5 ae_1 p_2$	$s_1\ \iota_6 ae_1^2 p_2 p_3 x$	$s_1\ \iota_7 ae_1^2 p_3 p_4 x$	$s_0\ \iota_8 b_5 b_6 b_7 x^3$	$s_0\ \iota_9$
4	$s_3\ \iota_1 ae_1^3 ho$	$s_3\ \iota_2 ae_2^2 p_1$	$s_3\ \iota_3 ae_2 e_4 p_1$	$s_3\ \iota_4 ae_4 p_1$	$s_2\ \iota_5 ab_8 e_1 p_2 x$	$s_2\ \iota_6 ab_8 e_1^2 p_2 x^2$	$s_2\ \iota_7 ab_8 e_1^2 p_4 x^2$	$s_1\ \iota_8 ae_2^2 e_4 p_5 p_6 p_7 x^2$	$s_0\ \iota_9 b_8 x$
5	$s_3\ \iota_1 ac_1 e_1^3 h^2 o$	$s_3\ \iota_2 ae_2^2 hp_1$	$s_3\ \iota_3 a^2$	$s_3\ \iota_4 ae_4 hp_1$	$s_3\ \iota_5 ae_1 e_6 p_2$	$s_3\ \iota_6 ac_1 e_1^2 e_6 p_2$	$s_3\ \iota_7 ac_1 e_1^2 e_6 p_4$	$s_2\ \iota_8 ab_9 e_2^2 e_4 p_6 x^3$	$s_1\ \iota_9 ae_6 p_8$
6	$s_3\ \iota_1 ae_1^2 h^3 o$	$s_3\ \iota_2 ac_2 e_2^2 h^2 p_1$	$s_4\ \iota_3 ac_4$	$s_3\ \iota_4 ac_4 e_4 h^2 p_1$	$s_3\ \iota_5 a^2$	$s_3\ \iota_6 ac_1 e_6 hp_2$	$s_3\ \iota_7 a^2 c_1$	$s_3\ \iota_8 ac_2 c_4 e_2^2 e_4 e_8 p_6$	$s_2\ \iota_9 ae_6 p_8$
7	$s_3\ \iota_1 ac_1 e_1^2 h^4 o$	$s_3\ \iota_2 ae_2 h^3 p_1$	$s_5\ \iota_3$	$s_3\ \iota_4 a^2 h^2$	$s_4\ \iota_5 a$	$s_3\ \iota_6 ac_1 e_6 h^2 p_2$	$s_4\ \iota_7 ac_1^2$	$s_3\ \iota_8 ac_2 c_4 e_8 hp_6$	$s_3\ \iota_9 ae_6 p_8$
8	$s_3\ \iota_1 ae_1 h^5 o^2$	$s_3\ \iota_2 ae_2 h^4 p_1$	$s_5\ \iota_3$	$s_4\ \iota_4 ah^2$	$s_5\ \iota_5$	$s_3\ \iota_6 ac_1 e_6 h^3 p_2$	$s_5\ \iota_7$	$s_3\ \iota_8 ac_2 c_4 c_8 e_8 h^2 p_6$	$s_3\ \iota_9 a^2$
9	$s_3\ \iota_1 ae_1 h^6 o^3$	$s_3\ \iota_2 ae_2 h^5 p_1$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5 c_6$	$s_3\ \iota_6 ac_1 c_6 e_6 h^4 p_2$	$s_5\ \iota_7 c_6$	$s_3\ \iota_8 a^2 c_2 c_4 h^2$	$s_4\ \iota_9 ac_6$
10	$s_3\ \iota_1 a^2 h^6$	$s_3\ \iota_2 ac_2 e_2 h^6 op_1$	$s_5\ \iota_3 c_2 o$	$s_5\ \iota_4 o$	$s_5\ \iota_5$	$s_3\ \iota_6 a^2 c_1 h^4$	$s_5\ \iota_7$	$s_4\ \iota_8 ac_2^2 c_4 h^2$	$s_5\ \iota_9$
11	$s_4\ \iota_1 ah^6$	$s_3\ \iota_2 a^3 h^6 p_1$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5$	$s_4\ \iota_6 ac_1 h^4$	$s_5\ \iota_7$	$s_5\ \iota_8$	$s_5\ \iota_9$
12	$s_5\ \iota_1$	$s_5\ \iota_2 ah^6$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5$	$s_5\ \iota_6$	$s_5\ \iota_7$	$s_5\ \iota_8$	$s_5\ \iota_9$
13	$s_5\ \iota_1 v^3 z$	$s_6\ \iota_2 aw^3$	$s_5\ \iota_3$	$s_5\ \iota_4$	$s_5\ \iota_5 v^3 z$	$s_5\ \iota_6 v^3 z$	$s_5\ \iota_7$	$s_5\ \iota_8$	$s_5\ \iota_9$
14	$s_6\ \iota_1 aw^2$	$s_6\ \iota_2 av^6 w^2 z^3$	$s_5\ \iota_3 v^4 z^2$	$s_5\ \iota_4 v^2 z$	$s_6\ \iota_5 aw^2$	$s_6\ \iota_6 aw^2$	$s_5\ \iota_7$	$s_5\ \iota_8 v^4 z^2$	$s_5\ \iota_9$
15	$s_6\ \iota_1 av^3 wz^3$	$s_6\ \iota_2 av^6 wz^3$	$s_6\ \iota_3 a^2 w^2$	$s_6\ \iota_4 aw$	$s_6\ \iota_5 av^2 wz^2$	$s_6\ \iota_6 av^4 wz^4$	$s_5\ \iota_7 v^5 z^5$	$s_6\ \iota_8 a^2 w^2$	$s_5\ \iota_9 v^2 z^2$
16	$s_6\ \iota_1 av^3 z^3$	$s_6\ \iota_2 av^6 z^3$	$s_6\ \iota_3 a^2 z^5$	$s_6\ \iota_4 az^5$	$s_6\ \iota_5 av^2 z^2$	$s_6\ \iota_6 av^4 z^4$	$s_6\ \iota_7 a^5$	$s_6\ \iota_8 a^2 z^7$	$s_6\ \iota_9 a^2$
17	$s_7\ \iota_1$	$s_7\ \iota_2$	$s_7\ \iota_3$	$s_7\ \iota_4$	$s_7\ \iota_5$	$s_7\ \iota_6$	$s_7\ \iota_7$	$s_7\ \iota_8$	$s_7\ \iota_9$

4.2 Phase I: Find the middle cell of a BFS spanning tree

For a given digraph-based P system, a (virtual) spanning tree is constructed by a standard BFS originated from the general, where the tree parent of each cell is one of its BFS parents (randomly chosen). Each cell keeps the track of its spanning tree parent and this is achieved by the use of cell IDs (unique identifier ID), e.g., i is the cell ID of σ_i.

Details of Phase I

Objective: The objective of Phase I is to find the middle cell, σ_m, and its height, $\texttt{height}_g(m)$.

Precondition: Phase I starts with the initial configuration of P system Π, described in Section 4.1.

Postcondition: Phase I ends when σ_m enters state s_5. At the end of Phase I, the configuration of cell $\sigma_i \in K$ is (Q, s_5, w_i, R), where $|w_i|_{\iota_i} = 1$; $|w_i|_a = 1$ and $|w_i|_h = 2 \cdot \texttt{height}_g(i)$, if $\sigma_i = \sigma_m$.

Description: We describe below the details of the BFS spanning tree construction and the propagation of the reflected symbol in the BFS tree. The symbol o, starting from the general, propagates from a tree parent to one of its children, as described in the FSSP solution for tree-based P systems (Section 3.2). Hence, the details of symbol o propagation are not given here.

- **The details of the BFS spanning tree construction:**
  A BFS starts from the general. When the search reaches cell σ_i, σ_i will send a copy of symbol b_i to all its neighbors (Rule 0.1 or 0.2).
  From the BFS, cell σ_i receives a copy of symbol b_j from each $\sigma_j \in \texttt{Pred}_g(i)$, where σ_j is a BFS dag parent of σ_i. Cell σ_i temporarily stores all of its BFS dag parents by transforming each received symbol b_j to symbol p_j (Rule 0.3). Note, σ_i will also receive a copy of symbol b_k from each $\sigma_k \in \texttt{Peer}_g(i) \cup \texttt{Succ}_g(i)$; however, σ_i will discard each received symbol b_k.
  Each cell selects one of its BFS dag parents as its tree parent. If cell σ_i has chosen σ_j as its tree parent, then σ_i will discards each p_k, where $\sigma_k \in \texttt{Pred}_g(i) \setminus \{\sigma_j\}$ (Rule 1.3). Additionally, σ_i will send a copy of symbol e_j to all its neighbors, which will be discarded by all σ_i's neighbors, except σ_j (Rule 1.1).
  Hence, in each cell σ_i, the multiplicity of symbol e_i will indicate the number of σ_i's tree children and symbol p_j will indicate that σ_j is the tree parent of σ_i; also, symbol p_j will later be used to propagate the reflected symbol back up the tree.
- **The details of reflected symbol propagation:**
  To replicate the propagation of a reflected symbol up the BFS tree, each internal cell of the BFS tree needs to check if the received a reflected symbol came from one of its BFS tree children.

Let σ_i be a BFS tree child of σ_j, where $|w_i|_{e_i} = 0$. Recall that, in such case, cell σ_i contains symbol p_j, where the subscript j is the ID of its BFS tree parent, and σ_j contains symbol e_j, such that $|w_j|_{e_j}$ is the number of σ_j's BFS tree children.
Guided by symbol p_j, σ_i sends symbol c_j to all its neighbors (Rule 3.9). Cell σ_j consumes a copy of symbol e_j with a copy of symbol c_j by Rule 3.3; σ_j cannot consume symbol e_j with symbol c_k, where $j \neq k$. If σ_j receives symbol c_j from all its BFS tree children, then all copies of symbol e_j will be consumed, i.e. $|w_j|_{e_j} = 0$.

Proposition 7 indicates the step in which the BFS reaches cell σ_i and σ_i receives symbol b_j from each $\sigma_j \in \texttt{Pred}_g(i)$. Proposition 8 indicates the step in which σ_i receives symbol e_i from its tree child.

Proposition 7. *Cell σ_i receives symbol b_j from each $\sigma_j \in \texttt{Pred}_g(i)$ at step $\texttt{depth}_g(i)$ and sends symbol b_i to all its neighbors at step $\texttt{depth}_g(i) + 1$.*

Proof. Proof by induction, on $d = \texttt{depth}_g(i) \geq 1$. At step 1, the general σ_g sends symbol b_g to all its neighbors by Rule 0.1. Hence, at step 1, each cell σ_k at depth 1 receives symbol b_g. Then, at step 2, by Rule 0.2, σ_k sends symbol b_k to each of its neighbors.

Assume that the induction hypothesis holds for each cell σ_j at depth d. Consider cell σ_i at $\texttt{depth}_g(i) = m + 1 = \texttt{depth}_g(j) + 1$. By induction hypothesis, at step $\texttt{depth}_g(j) + 1$, each $\sigma_j \in \texttt{Pred}_g(i)$ sends symbol b_j to all its neighbors. Thus, at step $\texttt{depth}_g(j) + 1 = \texttt{depth}_g(i)$, σ_i receives symbol b_j. At step $\texttt{depth}_g(i) + 1$, by Rule 0.2, σ_i sends symbol b_i to all its neighbors. □

Proposition 8. *Cell σ_i receives a copy of symbol e_i from each of its tree children at step $\texttt{depth}_g(i) + 3$.*

Proof. Assume that cell $\sigma_j \in \texttt{Succ}_g(i)$ has chosen σ_i as its tree parent. From Proposition 7, cell σ_j receives symbol b_i at step $\texttt{depth}_g(j) = \texttt{depth}_g(i) + 1$. According to the description, σ_j will send symbol e_i at step $\texttt{depth}_g(j) + 2$. Thus, σ_i will receive symbol e_i at step $\texttt{depth}_g(i) + 3$. □

Remark 1. From Proposition 8, σ_i receives symbol e_i from its tree child at step $\texttt{depth}_g(i) + 3$. If σ_i does not receive symbol e_i at step $\texttt{depth}_g(i) + 3$, then σ_i can recognize itself as a tree leaf and send a reflected symbol to its tree parent at step $\texttt{depth}_g(i) + 4$. That is, once a leaf cell is reached by the BFS, it will take three additional steps to send reflected symbol to its tree parent. Recall, in the FSSP algorithm for tree-based P systems, a leaf cell sends reflected symbol to its parent, one step after reached by the BFS. Thus, this FSSP algorithm for digraph-based P systems takes three additional steps to send the reflected symbol than the FSSP algorithm for tree-based P systems.

4.3 Phase II: Determine the step to enter the firing state

Similar to the Phase II described in Section 3.3, the firing order is broadcasted from the middle cell σ_m. The order is paired with a counter, which is initially set to the eccentricity of σ_m and decrements by one in each step of this broadcast operation.

Details of Phase II

Objective: The objective of Phase II is to determine the step to enter the firing state, such that during the last step of Phase II, i.e. the system's execution, all cells enter the firing state, simultaneously and for the first time.

Precondition: Phase II starts with the postcondition of Phase I, described in Section 4.2.

Postcondition: Phase II ends when all cells enter the firing state s_7. At the end of Phase II, the configuration of cell $\sigma_i \in K$ is $(Q, s_7, \{\iota_i\}, R)$.

Description: The order arrives in σ_i, along every shortest paths from σ_m to σ_i. Hence, to compute the correct step to enter the firing state, cell σ_i decrements, in each step, the sum of all received counter by the number of shortest paths from σ_m to σ_i and σ_i enters the firing state if the sum of all received counter becomes 0. The number of shortest paths from σ_m to σ_i is determined as follows. Cell σ_m sends a copy of symbol z. Each cell σ_i forwards symbol z, received from each $\sigma_j \in \texttt{Pred}_m(i)$. The number of shortest paths from σ_m to σ_i is the sum of all copies of symbol z that σ_i receives from each $\sigma_j \in \texttt{Pred}_m(i)$.

Let t be the the current counter and k be the number of shortest paths from σ_m to the current cell. In the FSSP solution for tree-based P systems, the condition for entering the firing state in the next step is when $t = 1$ (note $k = 1$). However, the FSSP solution, as implemented in this section, cannot directly detect if $t = k$, since $k \geq 1$ Instead, a cell enters the firing state after $t = 0$ is detected. Thus, the FSSP algorithm for digraph-based P systems requires one additional step in Phase II.

Theorem 4. *The synchronization time of the FSSP solution for digraph-based P systems is* $\texttt{ecc}(g) + 2 \cdot \texttt{ecc}(m) + 7$.

Proof. This FSSP algorithm for digraph-based P systems requires four additional overhead steps than the FSSP algorithm for tree-based P systems. Three of these four overhead steps are described in Remark 1 and the remaining overhead step is mentioned in Section 4.3. □

We end this section with a comment regarding improving the communication requirements of our FSSP solution. Currently, there may be an exponential number of broadcast objects generated since a given cell currently receives a copy of the counter from every possible shortest path from the middle cell. We can reduce number of broadcasted counters from an exponential to a polynomial as follows. Assume that, a counter, sent or forwarded from a cell, is annotated with the cell's ID. In Phase II, if a cell receives counter from its BFS tree neighbor (from a BFS

tree child for cells on the path from the general to the middle cell, otherwise from its original BFS tree parent), then it broadcasts the reduced-by-one counter, now annotated with its own ID, to all its neighbors. The total number of steps of this revised algorithm would still be the same as given in Theorem 4.

4.4 Empirical results

We also tested the improvement in running times over our previous FSSP algorithm on digraph-based P systems. The rate of improvement drops off as the number of edges increase over $n-1$, the size of trees of order n. But for several sparse digraph structured P systems the improvement is still worthwhile.

We did two tests suites; one for relatively small digraphs (illustrated in Figure 7) and one for larger digraphs as shown in Table 5. The graphs used in our empirical tests were generated using NetworkX [8].

For the statistics given in Table 5, we first generated connected random graphs of order n and size m. We then averaged over all possible locations for the general node. To model the parallel nature of P systems, we needed to generate a random BFS tree originating at the general. This was created by first performing a BFS from the general to constructing the BFS dag then randomly picking (for each non-general node) one parent within the dag structure as the parent for the BFS tree.

For this BFS tree, with e denoting the eccentricity of the general and r denoting the radius of the BFS tree, the "average gain" is the average difference of $3e - (e+2r)$ and the "average % gain" is the average of the $(3e-(e+2r))/(3e)$ values. From our empirical results, we can observe that the radius of the BFS spanning trees seems to be close to the actual radius of the given virtual communication graphs.

For the statistics given in the three dimensional plots of Figure 7 (generated using Gnuplot [20]), we generated 100 random connected (n, m)-graphs, for each order n, $20 \leq n \leq 40$, and size $m = (n-1)+2k$, where $0 \leq k \leq 20$. Note, the integer value of $2k$ represents the number of edges added to a tree. We then averaged over all possible general starting positions. The vertical axis is the average percentage speedup of our new algorithm over our previous synchronization algorithm. One can also observe from this plot, at least 20% improvements (i.e. reduction in number of steps needed to synchronize), is maintained for $k = 0$ (i.e. the graph is a tree). However, as the graphs become less sparse, the expected improvement drops to near zero, when as few as 40 edges are added to the trees. In general, for fixed k, the expected improvement in performance, for $(n, n+k)$ digraphs slightly increases as n increases. However, for fixed n, the expected improvement in performance drops drastically as k increases.

Table 5. Statistics for reduction in number of steps needed to synchronize on a few random (n, m)-graphs.

n	m	graph radius	avg tree radius	avgerage gain	avgerage gain %	n	m	graph radius	avg tree radius	avgerage gain	avgerage gain %
100	100	15	15.68	16.7	23.16	700	700	35	38.68	25.58	16.56
100	110	9	11.47	3.14	8.02	700	710	23	29.55	10.09	9.72
100	120	7	8.97	1.6	5.45	700	720	23	26.59	8.39	9.08
100	130	7	8.13	1.0	3.86	700	730	21	24.69	7.70	9.00
100	140	6	7.33	0.72	3.12	700	740	20	25.11	7.50	8.66
200	200	20	20.73	17.91	20.10	800	800	40	42.66	26.93	15.99
200	210	16	19.12	5.08	7.81	800	810	28	32.50	13.08	11.16
200	220	13	15.74	3.9	7.34	800	820	29	33.91	9.13	7.91
200	230	9	11.24	2.24	6.04	800	830	23	26.36	8.06	8.84
200	240	9	11.41	2.13	5.68	800	840	20	25.19	7.80	8.93
300	300	25	25.00	22.32	20.57	900	900	53	60.73	25.92	11.72
300	310	17	18.95	7.95	11.56	900	910	35	39.23	12.94	9.44
300	320	16	18.61	8.29	12.14	900	920	24	30.37	7.44	7.27
300	330	12	15.0	3.37	6.73	900	930	25	29.23	7.42	7.50
300	340	12	14.03	2.46	5.37	900	940	21	24.90	5.74	6.88
400	400	24	24.56	24.10	21.94	1000	1000	60	66.96	26.72	11.09
400	410	22	24.79	7.73	8.99	1000	1010	33	37.43	20.27	14.20
400	420	19	21.91	7.12	9.31	1000	1020	26	31.19	8.64	8.11
400	430	15	17.85	2.78	4.81	1000	1030	25	29.63	7.87	7.81
400	440	13	15.86	2.29	4.48	1000	1040	26	30.32	11.41	10.55
500	500	28	29.14	23.30	19.04	1000	1000	46	48.45	26.58	14.35
500	510	24	27.28	9.68	10.04	1000	1010	31	34.77	20.07	14.93
500	520	19	23.17	8.72	10.56	1000	1020	28	32.98	11.91	10.19
500	530	16	19.87	5.68	8.34	1000	1030	24	29.30	9.23	9.07
500	540	16	19.25	5.70	8.60	1000	1040	23	27.62	6.66	7.17
600	600	28	30.99	22.35	17.66	2000	2000	76	76.07	85.98	24.07
600	610	25	28.78	14.63	13.51	2000	2010	55	61.33	30.50	13.27
600	620	22	24.965	5.39	6.49	2000	2020	39	44.73	18.55	11.45
600	630	19	22.065	5.72	7.64	2000	2030	33	42.11	11.21	7.83
600	640	17	20.32	4.15	6.18	2000	2040	32	39.78	13.68	9.78

5 Conclusions and future works

In this paper, we explicitly presented an improved solution to the FSSP for tree-based P systems. We improved our previous FSP algorithm [5] by allowing the general to delegate a more central cell in the tree structure, as an alternative to itself, to send the final "firing" command. This procedure for trees-based P systems was extended to digraph-based P systems. Here we use a virtual spanning BFS tree (rooted at the general) in the digraph and use our tree-based middle-cell algorithm for that tree to improve the synchronization time. Alternatively, we would like to develop a way to compute a center of an arbitrary graph since the radius of the graph may be less than the radius of a particular BFS spanning tree. Thus this future work may possibly provide even more guaranteed improvements in synchronization time.

We summarize our work as follows. With e being the eccentricity of the general and r denoting the radius of the graph, where $e/2 \leq r \leq e$, we note the radius r' of the spanning BFS tree satisfies $e/2 \leq r \leq r' \leq e$. Thus, we have the following results:

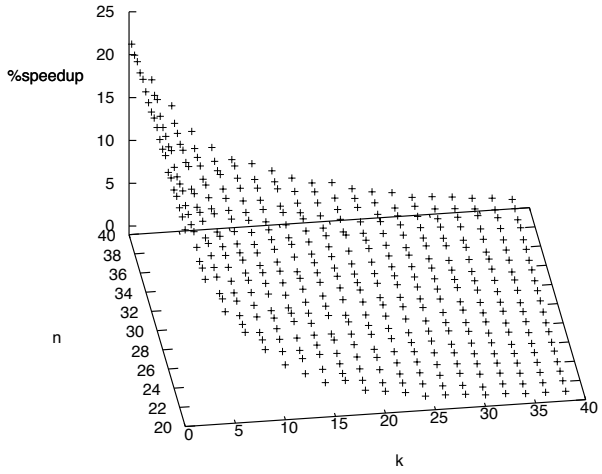


Fig. 7. Discrete 3-dimensional plot of expected synchronization improvements for a small range of random connected (n, m)-graph structures, with $m = (n-1) + k$ edges.

- If the membrane structure of a considered P system is a tree, then synchronization time is $e + 2r + 3$.
- If the membrane structure of a considered P system is a digraph, then synchronization time t is $e + 2r + 7 \leq t \leq 3e + 7$.

Our empirical work shows that the radius of the BFS spanning tree is often as small as the radius of its host graph and we expect, more often than not, the synchronization time to be closer to $e + 2r + 7$ than to $3e + 7$ for arbitrary digraph-based P systems.

Finally, we mention a couple open problems for the future. We would like a theoretical proof based on properties of random trees of why it seems that the our gain in performance is independent of the order of the trees considered. The current FSSP solution is designed for digraph-based P systems with *duplex* channels. Another remaining open problem is to obtain an efficient FSSP solution that synchronizes strongly connected digraphs using *simplex* channels.

Acknowledgments

The authors wish to thank Ionuţ-Mihai Niculescu for providing us with some early empirical statistics on random graphs and to acknowledge the University of Auckland FRDF grant 9843/3626216 to assist our research.

References

1. A. Alhazov, M. Margenstern, and S. Verlan. Fast synchronization in P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing*, LNCS 5391, pages 118–128. Springer, 2008.
2. R. Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1):22–42, 1967.
3. F. Bernardini, M. Gheorghe, M. Margenstern, and S. Verlan. How to synchronize the activity of all components of a P system? *Int. J. Found. Comput. Sci.*, 19(5):1183–1198, 2008.
4. A. Berthiaume, T. Bittner, L. Perkovic, A. Settle, and J. Simon. Bounding the firing synchronization problem on a ring. *Theor. Comput. Sci.*, 320(2-3):213–228, 2004.
5. M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. Faster synchronization in P systems. *International Journal of Natural Computing*, pages 1–15, 2011.
6. E. Goto. A minimal time solution of the firing squad problem. Course notes for Applied Mathematics 298, pages 52–59, Harvard University, 1962.
7. J. J. Grefenstette. Network structure and the firing squad synchronization problem. *J. Comput. Syst. Sci.*, 26(1):139–152, 1983.
8. A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug 2008.
9. M. Ionescu and D. Sburlan. On P systems with promoters/inhibitors. *J. UCS*, 10(5):581–599, 2004.
10. K. Kobayashi. The firing squad synchronization problem for a class of polyautomata networks. *J. Comput. Syst. Sci.*, 17(3):300–318, 1978.
11. E. F. Moore. The firing squad synchronization problem. In E. Moore, editor, *Sequential Machines, Selected Papers*, pages 213–214. Addison-Wesley, Reading MA., 1964.
12. F. R. Moore and G. G. Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.
13. Y. Nishitani and N. Honda. The firing squad synchronization problem for graphs. *Theor. Comput. Sci.*, 14:39–61, 1981.
14. G. Păun. Introduction to membrane computing. In G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors, *Applications of Membrane Computing*, pages 1–42. Springer-Verlag, 2006.
15. H. Schmid and T. Worsch. The firing squad synchronization problem with many generals for one-dimensional CA. In J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, editors, *IFIP TCS*, pages 111–124. Kluwer, 2004.
16. W. A. Stein et al. *Sage Mathematics Software (Version 4.6)*. The Sage Development Team, 2010. `http://www.sagemath.org`.
17. H. Umeo, N. Kamikawa, K. Nishioka, and S. Akiguchi. Generalized firing squad synchronization protocols for one-dimensional cellular automata—a survey. *Acta Physica Polonica B Proceedings Supplement*, 3(2):267–289, 2010.
18. A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.
19. E. W. Weisstein. Prüfer code, from MathWorld—a Wolfram web resource. `http://mathworld.wolfram.com/PrueferCode.html`, [Online; accessed 8-April-2011].
20. T. Williams, C. Kelley, and many others. Gnuplot 4.2: an interactive plotting program. `http://gnuplot.sourceforge.net/`, March 2009.

Toward a Self-replicating Metabolic P System

Giuditta Franco, Vincenzo Manca

Department of Computer Science,
University of Verona
{giuditta.franco, vincenzo.manca}@univr.it

Summary. This work concerns the synthesis of a 'minimal cell' by means of a P system, which is a distributed rewriting system inspired by the structure and the functioning of the biological cell. Specifically, we aim to define a dynamical system which exhibits a steady metabolic evolution, resulting in self-maintenance and self-reproduction. Metabolic P systems represent a class of P systems particularly promising to model a minimal cell in discrete terms, since they have already successfully modeled several metabolisms. The main further step is thus to find a simple way to obtain Metabolic P system self-replication.

This paper deals with ideas presented at the BWMC11 (held in Seville, Feb 2011) and opens a new trend in membrane computing, based on computational synthetic biology oriented applications of P systems modeling. The framework is here outlined, and some problems to tackle the synthesis of a minimal cell are discussed. Moreover, an overview of literature and a list of appealing research directions is given, along with several references.

1 Introduction

The idea of synthesizing a minimal cell by mathematical and engineered tools is not new in literature, namely there is a recent trend in synthetic biology which is aimed at building a synthetic endomembrane structure, whose compartments (usually formed by liposomes) contain the minimal and sufficient ingredients to perform the basic function of a biological cell (essentially self-maintenance and self-reproduction).

Such an interest originates from the old wondering about "what is life?", and specifically from the question "*how* was possible for a primitive chemical system to evolve through levels of increasing complexity from disordered and unstructured primordial soup to the cellular life as we know it?" [17]. In scientific research, the bottom-up approach, which looks for a plausible process leading from simple molecules to more complex ones, to protocells, and finally to living cells, has still many open questions, although some progress has been done in our knowledge of prebiotic chemistry [14]. An alternative approach, called top-down, focuses on the

synthesis of minimal forms of life starting from our knowledge of modern cells, that is, from general principles of structure and function organization (matter conservation, anabolism and catabolism, species distribution, enzymatic control, autopoiesis). This has an experimental counterpart, and it can be classified as a constructivist approach for scientific knowledge, according to Feynmans famous motto "What I cannot create, I do not understand".

In principle, several implementations of minimal life are possible [19], namely primitive cells, minimal cells, bioreactors, molecular robots (soft-robots) [14]. Current experimental strategies consider synthetic cells as systems having two main components: compartments (i.e., lipid vesicles) and their content (biomacromolecules, such as DNA, RNA, enzymes, ribozymes, PNA, ribosomes, catalytic peptides) [14, 15]. In this context, aggregation phenomena need to be reproduced in laboratory and a preliminary simulation in silico helps to set the quantity range of RNA-polimerasi and RNA-sintasi, in order to get an efficient synchronization of self-mantainance and self-replication [3].

2 Some models

In the cell, molecules react together according to their biochemical reactivity and environmental conditions, giving rise to complex molecules starting from simpler ones. Also, a higher chemical complexity (usually referred to as "supramolecular chemistry") appears as the result of self-organization of molecules into structures (membranes) and oscillating reactions such as auto-catalytic networks within micro-compartments [15].

Among the most active groups working on creating living cells in the laboratory, we recall David Deamer at the University of California, Jack Szostak at Harvard, Tetsuya Yomo at the Osaka University, Steen Rasmussen at the FLinT (Southern Denmark University). Besides we mention the notable research, both on the construction of self-reproducing vesicles and on synthetic minimal cells, started about twenty years ago in the Luisi's group at the ETH (Zurich). It roots in the concept of *autopoiesis*, the theoretical framework that guides the construction of minimal living cells and accounts for the dynamical process at the basis of living entities [15]. The notion of *autopoietic cell* dates back to the work of H. R. Maturana and F. J. Varela in the seventies [11]. It essentially requires a shell/membrane composed by *i)* building blocks L (representing the lipids and the proteins of cell membranes), that eventually decay to a waste product W, and *ii)* an internal metabolism, a black box E (representing the cellular genetic/metabolic network), able to both generate blocks L (from precursors P entering the membrane from the environment) and maintain a transformation of metabolites Q (arriving from outside) that produces and expels waste product Z. According to this scheme, P and Q are the basic nutrients for cell growth, W and Z the waste materials.

In [15] it has been shown that a supramolecular assembly of L molecules can grow and duplicate at the expenses of matter P from outside without any internal

metabolism. However, if we impose to have an autopoietic mechanism, based on a minimal, existent DNA/RNA/enzyme genetic/metabolic network E, then minimal cells exhibiting living properties (self-mantainance, self-reproduction, and possibility to evolve) have a minimal number of genes (a number between 200 and 300, according to results from literature in comparative genomics), enzymes, robosomes, tRNAs and low molecular weight compounds [1, 15]. In this context, protein synthesis is one of the key function for a living cell. The expression of functional proteins inside lipid vesicles by using a minimal set of enzymes, tRNAs and ribosomes, was also investigated in [15] at the aim of constructing continuous models of functional cells, while an efficient protein-synthesizing system was developed in [13].

In [3] a kinetic model of (autopoietic) ribocell was built by means of a differential equation system, where variations of metabolite and lipid concentrations, as well as membrane volume variations, are established by modeling processes such as RNA strands replication, catalyzed by polymerase ribozyme, pairings of RNA-polymerase and RNA-sintase, and conversions of precursors into membrane lipids, catalyzed by ribozymes. The time evolution is deterministic rather than stochastic, by assuming that in average different membranes have the same time behavior. The expansion measure of membrane surfaces is considered, in such a way that self-replication of the whole cell (in two daughters) is assumed as soon as the membrane surface reaches the area sufficient to form two spherical membranes. This model resulted in synchronized genomic duplication and cell replication, with the kinetic values within ranges suggested by the literature. According to the simulations reported in [3], (at room temperature) cell division occurs every 26.6 days, and may be speeded up by increasing the temperature (for example, up to 42°C). The goal of our research is to reproduce a similar autopoietic deterministic system in discrete terms, where biomolecules are represented by multisets of objects, membranes are compartments where rewriting rules are distributed to work in parallel, and the computation is the dynamics observed in a cell at a "suitable" level of abstraction. von Neumann first conceived a self-replicating computational model, by pioneering cellular automata (CA) able to self-replicate [20], but that "mitosis process" was not supposed to be synchronized with any internal metabolism or with other properties typical of the biological cell.

3 Main questions

According to the top-down approach, building a synthetic cell by means of a computational model is in itself a way to understand (or at least to get more information on) the basic concepts of living systems and of their parts.

A main question here is: what are the minimal components, the simplest form of machinery, to get 'biological universality' (behaviors typical of life)? We can say that the minimal number of life criteria is three: self-mantainance, self-reproduction, and evolution capability [7]. Then, a metabolism (internal dynamics) in compart-

ments has to be realized (self-mantainance), together with a simultaneous replication of main internal components and of all membranes (self-reproduction). This is driven by genomic information, by means of gene expression, which gives rise to (structural or enzymatic) proteins, able to perform functions (such as catalyzing biochemical reactions occurring in metabolism).

From a logical viewpoint, Is it necessary to have the genetic/regulative mechanism at the basis of metabolism? Or maybe the presence of ribozomes (naturally self-replicating substances), RNA polymerase and nucleotides, would be enough to have an RNA-based autocatalyst living system? According to the RNA world hypothesis [5], a set of rybozymes is actually sufficient, because RNA can both store information like DNA and act as an enzyme like protein. In this perspective, DNA polymer is just a product of evolution to have redundancy and robustness to errors, including point mutations. In modern cells indeed, DNA, through its greater chemical stability, took over the role of data storage, while proteins, which are more flexible in catalysis through the great variety of amino acids, play only the role of specialized catalytic RNA molecules. The presence of both genomes and ribozymes is therefore redundant to have a simplest self-replicating metabolic system [6], though most likely an independent storage mechanism is required for systems which adapt to the environment. To have a system with the capability to evolve by adaption, sensitivity and adaption to the environment need to be taken into account in the model, by analyzing the exchange of matter with the environment, and the reaction of the system dynamics to environment changes.

Overall, nature exhibits the two levels, informational (genes) and functional (enzymes) - are they necessary to perform an efficient mitosis, or to realize the cell (Darwinian) evolution? A possible answer is that in cells of complex organisms, which need to store more information and for a longer time, the stability of genomic molecules make their existence necessary to have life. In this case, the genomic level would have turned out necessary in the evolution in order to allow a major and more structured complexity of organisms.

4 Our approach

A self-replicating metabolic system requires a synchronization of its internal dynamics in such a way that the metabolic activity is maintained, while the system exchanges matter with the environment, grows, and replicates its own membrane structure together with the contained metabolic processes. We aim at modeling such a dynamical system by a P system [16], that is, by a computational model inspired by the cell. This approach seems the most natural to reproduce in silico what is observed in the cell. Hence, this research may be framed in a context of membrane computing models, and aims at building a self-replicating metabolic membrane system where molecular and cellular peculiarities are represented in symbolic and algorithmic terms.

A similar work has been developed in [18], where a self-replicating membrane system has been exhibited, which initiates with a process of self-inspection, then

copies the membrane contents (objects and rules), incrementally composes the genome and finally create the mother cell outside the current skin membrane. With respect to our goal, in [18] there is not any metabolism in the cell, and the rules are inspired by artificial rather than biological systems.

This subject had already attracted attention in [2], where, however, entire membrane systems were replicated in one macro-step. A closer look to biology may be found in [12], where a "Dogmatic P system" (inspired by the central dogma of molecular biology), which exhibits transduction and transcription processes in the nucleus, is proved to be universal.

A P system is a multi-compartment structure realizing a parallel, distributed, object multiset rewriting system (for more details see `http://ppage.psystems.eu/`). A metabolic P system (shortly, MP system [10]) is essentially a multiset grammar where multiset transformations are regulated by state functions (called regulators), whose values (at each step) represent the fluxes associated to the rewriting rules [8]. Once we know the regulators, a deterministic Markovian dynamics of such systems may be observed as time series of the substances. On the other hand, there are theories and algorithms developed in the framework of Metabolic P systems, which allow to compute regulators starting from observed time series [9].

We would like to keep our self-duplicating metabolic P model as simple as possible - we do not use extra features (such as priorities, polarization, fluxes, probability) if they are not necessary or biologically motivated.

4.1 A research plan

The point is defining a Metabolic P model, in order to understand which are the general rules and regulations on which the synchronization of genomic duplication and membrane reproduction are based.

One cell divides to produce two genetically identical cells. Eukaryotic cells include a variety of membrane-bound structures, collectively referred to as the endomembrane system. Nuclear division is often coordinated with cell division. As a first approximation, we assume to have only two, nested (external and nuclear) membranes $[_0[_1]_1]_0$.

A possible initial conguration (having all genes in the nucleus) includes a multiset of objects g representing genes, r objects representing enzymes, one ribozyme t, and some metabolites (including lipids). Environment is assumed to have "sufficient" resources m.

The goal is to exchange objects with the environment, while metabolites growing, up to reach an "approximatively" double amount they had initially. The concept of matter duplication should be further defined, for example by asking for having an amount which is double than it was initially within a certain range of error, but in this first attempt we leave it undetermined.

Simultaneously, enzymes are produced from genes, for feeding reactions, which guarantee an internal metabolic dynamics of the system, and generate new membranes. In this respect, let us list in the following paragraph some biologically motivated guidelines to set up our model.

Genes replicate (by polymerase) and produce enzymes (by ribozymes). They do not move across the (nuclear) membrane. Ribozymes are self-producing (by means of genes). Enzymes catalyze and allow the application of rewriting rules, in fact metabolites transform by means of rules. Lipids, produced by the metabolism, generate membranes, when present in sufficient amount. In order to account for the size of a membrane, we assume that more external membranes are, and more lipids are necessary to generate a new copy of them.

The rewriting rules will be meant to realize three main processes: enzyme production, metabolism, and liposome production.

Enzyme production. Nuclear transcription of the genes is realized by means of the ribozyme (we can assume to have only one ribozyme, able to perform transcription) and some nucleotides. It produces the enzyme polymerase in the nucleus, and three enzymes r, r', r'' in the region of the external membrane. In the most internal membrane, polymerase is employed to duplicate the genes by using nucleotides - maybe this could be done more realistically with string rewriting rules.

Metabolism. Matter is taken from outside by both membranes. Enzymes produced in the nucleus (by means of the ribozyme) catalyze transformation reactions: specifically, the rule catalyzed by r increases the quantity of the ribozyme itself at the expenses of matter arriving from outside; the rule catalyzed by r' produces lipids able to form the nuclear membrane, and the rule catalyzed by r'' produces lipids able to form the external membrane. Of course we have also matter which does not need to be duplicated, just working as fuel (coming from outside) and as garbage (expelled outside). In a further refinement of the model enzyme degradation should be also considered.

Liposome production. Lipids transform into membranes (liposome organization), representing their organization in structures as vesicles.

Once given the rules, the choice of an application strategy needs some discussion. Even if a non-deterministic or probabilistic evolution of a proto-cellular system could be interesting to study, here we intend to reproduce the deterministic behaviour typically observed in cell replication. Presumably we cannot avoid to use flux functions associated to the rules (as metabolic systems do), for example to regulate the entrance of objects from outside. Maximal parallelism would imply that the cell gets in one first step the infinite resources we have in the environment! Fluxes instead would allow us to modulate any reaction according to the system state.

As an initial set up, a few equations/constraints need to be given in order to impose that there exists a moment k, where the genes and the ribozymes are approximatively double than they were initially, and the lipids are in sufficient amount to form a second copy of the membranes. We assume that reaching such a state will be enough to have two copies of the initial cell.

5 Future Work

Regarding future activities of this research, it is plenty of ideas and dreams. Once we will have an MP system self-replicating, while exchanging matter with the environment to keep its internal metabolic dynamics on, both the role of energy in such an exchange and a form of adaption to the environment should be studied [14], by analyzing the consequent reactions of the system to different (even energetic) stimuli. Receptivity and reactivity should be investigated to better understand the robustness of the single cell and of cell networks. Communication and interaction among (synthetic and/or real) cells is a crucial task [4], for example to model morphogenesis (e.g., embryogenesis) and tissue organization. From the viewpoint of a tissue system, the process of mitosis of each single cell is limited in time, single healthy cells do not live forever but tissue do, while new cells rise and old ones die. Tissues keep alive under certain boundaries (density, dimension) while single cells produce new cells and eventually die: the cellular and tissue systems have a quite different dynamics and functioning, even if tightly interrelated. In [15] simple autopoietic systems are modeled by vesicles populations; it is shown that simple vesicles may grow and divide according to physical laws, also revealing an unexpected pattern consisting in the conservation of the average size in a population of self-reproducing vesicles.

Phenomena such as cell differentiation and speciation are fundamental to understand and better control many processes of biomedical interest. For example, embryonic cells are interesting as they have illimitable replicative power and the ability to generate any type of tissue, a property they have in common with stem cells. On the other hand, non-controlled proliferation and differentiation of stem cells often denote presence of cancer. Cell migration can be also involved in such kinds of processes, and in our research a way to represent both molecular and cellular migration in the context of P systems should be found. In this framework we should start by modeling the concept of biological gradient, maybe by means of nested membrane localization. Finally, cell Darwinian evolution of synthetic cells could give interesting insights on several controversial issues in population genetics evolution theories (such as the importance of the chance in evolutionary transformations, known as genetic drift).

Aknowledgments

We would like to thank Natasha Jonoska for her helpful comments and stimulating suggestions about the presentation of this work at the Brainstorming on membrane computing held in Seville in February 2011. On the occasion of the meeting, the first author gratefully received interesting, specific highlights by many colleagues, with whom there were fruitful discussions. In particular, she wishes to thank very much George Păun, Alfonso Rodrigez Paton, Mario Pérez-Jiménez, Marian Gheorghe, and José Maria Sempere.

References

1. C. Chiarabelli, P. Stano, P. L. Luisi, *Chemical approaches to synthetic biology*, Current Opinion in Biotechnology 20(4): 492-497, 2009.
2. E. Csuhaj-Varjú, A. Di Nola, Gh. Păun, M. J. Pérez-Jiménez, G. Vaszil, *Editing configurations of P systems*, Third Brainstorming Week on Membrane Computing, pp 131-154, 2004.
3. P. Della Gatta, F. Mavelli, *Ribocell modeling*, Wivace 2009, pp 55-64.
4. P. M. Gardner, K. Winzer, B. G. Davis, *Sugar synthesis in a protocellular model leads to a cell signaling response in bacteria*, Natural Chemistry 1, pp 377–383, 2009.
5. W. Gilbert, *Origin of life: The RNA world*, Nature 319:618, 1986.
6. G. F. Joyce, L. E. Orgel, *The RNA world* (Eds R. Gesteland, T. R. Cech, J. F. Atkins], pp 49-77, Cold Spring Harbor Laboratory Press, New York, 1999.
7. P. L. Luisi, *About various definitions of life*, Origins of Life and Evolution of the Biosphere 28, pp 613-622, 1998.
8. V. Manca, *Fundamentals of metabolic P systems*, In: Gh. Paun, G. Rozenberg, A. Salomaa (eds.), Handbook of Membrane Computing, Chapter 19, Oxford University Press, 2009.
9. V. Manca, *Metabolic P dynamics*, In: Gh. Paun, G. Rozenberg, A. Salomaa (eds.), Handbook of Membrane Computing, Chapter 20, Oxford University Press, 2009.
10. V. Manca, *Metabolic P systems*, Scholarpedia, 5(3):9273, 2010.
11. H. R. Maturana, F. J. Varela, *Autopoiesis and cognition: the realization of the living.* Reidel, Dordrecht, 1980.
12. J. M. Sempere, *"Dogmatic" P systems*, Eighth Brainstorming Week on Membrane Computing, pp 291- 300, 2010.
13. Y. Shimizu, A. Inoue, Y. Tomari, T. Suzuki, T. Yokogawa, K. Nishikawa, T. Ueda, *Cell-free translation reconstituted with purified components*, Nature Biotechnology 19, pp 751- 755, 2001.
14. P. Stano, *Cellule artificiali: dall'attuale quadro teorico-sperimentale al loro uso come robot molecolari*, Wivace 2009, pp193-198.
15. P. Stano, P. L. Luisi, *Chemical approaches to synthetic biology: from vesicles self-reproduction to semi-synthetic minimal cells*, Proc. of the Alife XII Conference, Odense, Denmark, pp 147-153, 2010.
16. G. Păun, *Membrane computing. An introduction.* Springer, 2002.
17. J. W. Szostak, D. P. Bartel, P. L. Luisi, *Synthesizing life*, Nature 409: 387-390, 2001.
18. C. Teuscher, *From membranes to systems: Self-configuration and self-replication in membrane systems*, Biosystems 87, pp 101- 110, 2007.
19. C. Venter, *Creation of a bacterial cell controlled by a chemically synthesized genome*, Science 329 (5987): 52-56, 2010.
20. J. von Neumann, *Theory of self-reproducing automata*, University of Illinois Press, Urbana,, Illinois, 1966.

Implementing Local Search with Membrane Computing

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

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

Summary. Local search is currently one of the most used methods for finding solution in real-life problems. In this paper we present an implementation of local search with Membrane Computing techniques applied to the N-queens problem as a case study. A CLIPS program inspired in the Membrane Computing design has been implemented and several experiments have been performed.

1 Introduction

Searching is on the basis of many processes in Artificial Intelligence. The key point is that many real-life problems can be settled as a *space of states*: a *state* is the description of the world in a given instant (expressed in some language) and two states are linked by a *transition* if the second state can be reached from the previous one by applying one *elementary operation*. By using these concepts, a directed graph where the nodes are the states and the edges are the actions is considered. Given a starting state, a sequence of transitions to one of the final states is searched.

By using this abstraction, searching methods have been deeply studied by themselves, forgetting the real-world problem which they fit. The studies consider aspects as the completeness (if the searching method is capable of finding a solution if it exists), complexity in time and space, and optimality (if the found solution is *optimal* in some sense). By considering the searching tree, where the nodes are the states and the arcs are the transitions, classical search has focused on the order in which the nodes of the tree should be explored. In this classical search two approaches are possible: the former is *blind search*, where the search is guided only by the topology of the tree and no information is available from the states; the latter is called *informed search* and some information about the features of the nodes is used to define a *heuristics* to decide the next node to explore.

Searching problems have been previously studied in the framework of Membrane Computing. In [4], a first study on depth-first search on the framework of

Membrane Computing was presented. In this paper we go on with the study of searching methods in Membrane Computing by exploring local search.

The paper is organized as follows: First we recall some basic definitions on local search. Then we remember the problem used as a case study: the N-queens problem, previously studied in the framework of Membrane Computing in [3]. Next we provide some guidelines of the implementation of local search our case study and some experimental results. The paper finishes with some final remarks.

2 Local Search

Classical search algorithms explore the space of states systematically. This exploration is made by keeping one or more paths in memory and by recording the alternatives in each choice point. When a final state is found, the path, that is, the sequence of *transitions*, is considered the solution of the problem. Nonetheless, in many problems, we are only interested in the found state, not properly in the path of transitions. For example, in job-shop scheduling, vehicle routing or telecommunications network optimization, we are only interested in the final state (a concrete disposition of the objects in the world), not in the way in which this state is achieved.

If the sequence of elementary transitions is not important, a good alternative to classical searching algorithms is *local search*. This type of search operates using a single state and its set of neighbors. It is not necessary to keep in memory how the current state has been obtained.

Since these algorithms do not explore systematically the states, they do not guarantee that a final state can be found, i.e., they are not complete. Nonetheless, they have two advantages that make them interesting in many situations:

- Only a little piece of information is stored, so a very little memory (usually constant) is used.
- These algorithms can often find a reasonable solution in extremely large space of states where classical algorithms are unsuitable.

The basic strategy in local search is considering a current state and, if it is not a final one, then to move to one of its neighbors. This movement is not made randomly. In order to decide where to move, in local search a *measure of goodness* is introduced. In this way, the movement is performed towards the best neighbor. It is usual to represent the *goodness* of a state as its height in some geometrical space. In this way, we can consider a landscape of states and the target of the searching method is to arrive to the *global maximum*. This metaphor is useful to understand some of the drawbacks of this method. Different situations as *flat regions*, where the neighbors are as good as the current state, or *local maximum* where the neighbors are worse than the current state, but it is not a *global maximum*. A deep study of local search is out of the scope of this paper[1].

[1] Further information can be obtained in [7].

In this paper we will only consider its basic algorithm: Given a *set of states*, *a movement operator* and *a measure to compare states*

0.- We start with a state randomly chosen

1.- We check if the current state is a final one

1.1- If so, we finish. The system outputs the current state.

1.2- If not, we look for a movement which reaches a *better* state.

1.2.1.- If it exists, we randomly choose one of the possible movements. The reached state becomes the *current state* and we back to 1.

1.2.2.- If it does not exist, we go back to 0

3 The *N*-queens Problem

Along this paper we will consider the N-queens problem as a case study. It is a generalization of a classic puzzle known as the 8-queens puzzle. The original one is attributed to the chess player Max Bezzel and it consists on putting eight queens on an 8×8 chessboard in such way that none of them is able to capture any other using the standard movement of the queens in chess, i.e., only one queen can be placed on each row, column and diagonal line.

In [3], a first solution to the N-queens problem in Membrane Computing was shown. For that aim, a family of deterministic P systems with active membranes was presented. In this family, the N-th element of the family solves the N-queens problem and the last configuration encodes *all* the solutions of the problem.

In order to solve the N-queens problem, a truth assignment that satisfies a formula in conjunctive normal form (CNF) is searched. This problem is exactly SAT, so the solution presented in [3] uses a modified solution for SAT from [6]. Some experiments were presented by running the P systems with an updated version of the P-lingua simulator [2]. The experiments were performed on a system with an Intel Core2 Quad CPU (a single processor with 4 cores at 2,83Ghz), 8GB of RAM and using a C++ simulator under the operating system Ubuntu Server 8.04. According to the representation in [3], the 3-queens problem is expressed by a formula in CNF with 9 variables and 31 clauses. The *input multiset* has 65 elements and the P system has 3185 rules. Along the computation, $2^9 = 512$ elementary membranes need to be considered in parallel. Since the simulation was carried out on a uniprocessor system, these membranes were evaluated sequentially. It took 7 seconds to reach the halting configuration. It is the 117-th configuration and in this configuration one object `No` appears in the environment. As expected, this means that we cannot place three queens on a 3×3 chessboard satisfying the restriction of the problem.

In the 4-queens problem, we try to place four queens on a 4×4 chessboard. According to the representation, the problem can be expressed by a formula in CNF with 16 variables and 80 clauses. Along the computation, $2^{16} = 65536$ elementary membranes were considered in the same configuration and the P system has 13622 rules. The simulation takes 20583 seconds (> 5 hours) to reach the halting configuration. It is the 256-th configuration and in this configuration one object `Yes`

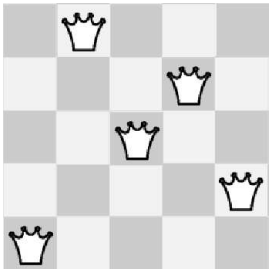


Fig. 1. Five queens on a board

appears in the environment. This configuration has two elementary membranes encoding the two solutions of the problem (see [3] for details).

In [4], a study of depth-first search in Membrane Computing was presented. The case study was also the N-queens problem. An *ad hoc* CLIPS program was written based on Membrane Computing design. Some experiments were performed on a system with an Intel Pentium Dual CPU E2200 at 2,20 GHz, 3GB of RAM and using CLIPS V6.241 under the operating system Windows Vista. Finding one solution took 0,062 seconds for a 4×4 board and 15,944 seconds for a 20×20 board.

4 A P system Family for Local Search

In this section we give an sketch of the design of a P system family which solves the N-queens problem by using local search, $\mathbf{\Pi} = \{\Pi(N)\}_{N \in \mathbb{N}}$. Each P systems $\Pi(N)$ solves the N-queens problem in a non-deterministic way, according to the searching method. The membrane structure does not change along the computation and we use electrical charges on the membranes as in the model of active membranes.

One *state* is represented by a $N \times N$ chess board where N queens have been placed. In order to limit the number of possible states, we will consider an important restriction: we consider that there is only one queen in each column and in each row. By using this restriction, we only need to check the diagonals in order to know if a board is a solution to the problem or not.

These boards can be easily represented with P systems. For $\Pi(N)$, we consider a membrane structure which contains N elementary membranes and N objects y_i, $i \in \{1, \ldots, N\}$ in the skin. By using rules of type $y_i\,[\,]_j \rightarrow [y_i]_j$ the objects y_i are non-deterministically sent into the membranes and the object y_i inside a membrane with label j will be interpreted as a queen placed on the row i of the column j. For example, the partial configuration $[\,[y_1]_1\ [y_5]_2\ [y_3]_3\ [y_4]_4\ [y_2]_5\,]$ is a membrane representation of the board in Figure 1.

In order to know if one state is better than another, we need to consider a *measure*. The natural measure is to associate to any board the number of *collisions*

[8]: *The number of collisions on a diagonal line is one less than the number of queens on the line, if the line is not empty, and zero if the line is empty. The sum of collisions on all diagonal lines is the total number of collisions between queens.* For example, if we denote by d_p the descendant diagonal for squares (i, j) where $i+j = p$ and by u_q the ascendant diagonal for squares (i, j) where $i-j = q$, then the board from the Figure 1 has 3 collisions: 2 in u_0 and 1 in d_7. This basic definition of *collisions of a state* can be refined of a Membrane Computing algorithm. As we will see below, in order to compare two boards, it is not important the exact amount of collisions when they are greater than 3.

Other key definitions in the algorithm are the concepts of *neighbor* and *movement*. In this paper, a movement is the interchange of columns of two queens by keeping the rows. In other words, if we have one queen at (i, j) and another in (k, s), after the movement these queens are placed at (i, s) and (k, j). It is trivial to check that, for each movement, if the original board does not have two queens on the same column and row, then the final one does not have it. The definition of neighbor depends on the definition of movement: the state s_2 is a neighbor of state s_1 if it can be reached from s_1 with one movement.

According with these definitions, the local search algorithm for the N-queens problem can be settled as follows:

0.- We start with a state randomly chosen
1.- We check if the number of collisions of the current state is zero
  1.1- If so, we finish. The halting configuration codifies the solution board.
  1.2- If not, we look for a movement which reaches a state which decrease the number of collisions.
    1.2.1.- If it exists, we randomly choose one of the possible movements. The reached state becomes the *current state* and we back to 1.
    1.2.2.- If it does not exist, we go back to 0

At this point, three basic question arise from the design of Membrane Computing: (1) how the number of collisions of a board is computed? (2) how a *better* state is searched? and (3) how a movement is performed?

4.1 Computing Collisions

The representation of a board $N \times N$ is made by using N elementary membranes where the objects $y_1, \ldots, y_N$ are placed. These N elementary membranes, with labels $1, \ldots, N$ are not the unique elementary membranes in the membrane structure. As pointed above, in an $N \times N$ board, there are $2N-1$ ascendant and $2N-1$ descendant diagonals. We will denote the ascendant diagonals as $u_{-N+1}, \ldots, u_{N-1}$, where the index p in u_p denotes that the diagonal corresponds to the squares (i, j) with $i-j = p$. Analogously, the descendant diagonals are denoted by $d_2, \ldots, d_{N+N}$ where the index q in d_q denotes that the diagonal corresponds to the squares (i, j) with $i + j = q$.

Besides the N elementary membranes with labels $1, \ldots, N$ for encoding the board, we will also place $4N - 2$ elementary membranes in the structure, with

labels $u_{-N+1}, \ldots, u_{N-1}, d_2, \ldots, d_{N+N}$. These membranes will be used to compute the collisions.

Bearing in mind the current board, encoded by membranes with an object $[y_i]_j$, we can use rules of type $[y_i]_j \rightarrow d_{i+j}u_{i-j}$. These rules are triggered in parallel and they produce as many objects d_q (resp. u_p) as queens are placed on the diagonal d_q (resp. u_p).

Objects d_q and u_p are sequentially sent into the elementary membranes labelled by d_q and u_p. In a first approach, one can consider a counter z_i which evolves to z_{i+1} inside each elementary membrane when an object d_q or u_p is sent in. By using this strategy, the index of i of z_i denotes how may objects have crossed the membrane, or in other words, how many queens are placed on the corresponding diagonal.

This strategy has an important drawback. In the worst case, if all the queens are placed on the same diagonal, at least N steps are necessary in order to count the number of queens in each diagonal. In our design, this is not necessary. As we will see, we only need no know if the number of queens in each diagonal is 0,1,2, or more than 2. Due to the parallelism of the P systems, this can be checked in a constant number of steps regardless the number of queens.

Technically, after a complex set of rules where the electrical charges are used to control the flow of objects, each membrane d_q sends to the skin a complex object of type $d_q(DA_q, DB_q, DC_q, DD_q)$, where $DA_q, DB_q, DC_q, DD_q \in \{0, 1\}$ codify on the number of queens on the diagonal (for u_p the development is analogous, with the notation $u_p(UA_p, UB_p, UC_p, UD_p)$). We consider four possibilities:

- $d_q(1, 0, 0, 0)$. The 1 in the first coordinate denotes that there is no queens placed on the diagonal and the diagonal is ready to receive one queen after a movement.
- $d_q(0, 1, 0, 0)$. The 1 in the second coordinate denotes that there is one queen placed on the diagonal. This diagonal does not contain collisions but it should not receive more queens.
- $d_q(0, 0, 1, 0)$. The 1 in the third coordinate denotes that there are two queens placed on the diagonal. This diagonal has one collision which can be solved with a unique appropriate movement.
- $d_q(0, 0, 0, 1)$. The 1 in the fourth diagonal denotes that there are more than two queens placed on the diagonal. This diagonal has several collisions and it will have at least one collision even if one movement is performed.

Bearing in mind if a diagonal is ready to receive queens (0 queens) or it needs to send queens to another diagonal, we can prevent if a movement produces an improvement in the whole number of collisions before performing the movement. We do not need to perform the movement and then to count the number of collisions in order to know if the movement decreases the number of collisions.

Firstly, let us consider two queens placed on the squares (i, j) and (k, s) on the same ascendant diagonal, i.e., $i - j = k - s$. We wonder if the movement of interchanging the columns of two queens by keeping the rows will improve the

total number of collisions. In other words, we wonder if removing the queens from (i, j) and (k, s) putting them in (i, s) and (k, j) improves the board.

In order to answer this question we will consider:

- $u_{i-j}(0, 0, UC_{i-j}, UD_{i-j})$: The ascendant diagonal u_{i-j} has at least 2 queens, so the two first coordinates are 0.
- $d_{i+j}(0, DB_{i+j}, DC_{i+j}, DD_{i+j})$ and $d_{k+s}(0, DB_{k+s}, DC_{k+s}, DD_{k+s})$: The descendent diagonals d_{i+j} and d_{k+s} have at least 1 queen, so the we first coordinate is 0.

It is easy to check that the reduction in the whole amount of collisions produced by the removal of the queens from the squares (i, j) and (k, s) is

$$(2 \cdot UC_{i-j}) + (3 \cdot UD_{i-j}) + DB_{i+j} + (2 \cdot DC_{i+j}) + (3 \cdot DD_{i+j}) + DB_{k+s} +$$

$$(2 \cdot DC_{k+s}) + (3 \cdot DD_{k+s}) - 3$$

Analogously, in order to compute the augmentation in the number of collisions produced by the placement of two queens in the squares (i, s), (k, j) we will consider the objects $d_{i+s}(DA_{i+s}, DB_{i+s}, DC_{i+s}, DD_{i+s})$, $u_{i-s}(UA_{i-s}, UB_{i-s}, UC_{i-s}, UD_{i-s})$ and $u_{k-j}(UA_{k-j}, UB_{k-j}, UC_{k-j}, UD_{k-j})$.

By using this notation, it is easy to check that the augmentation in the number of collisions is $4 - (DA_{i+s} + UA_{i-s} + UA_{k-j})$.

The movement represents an improvement in the general situation of the board if the reduction in the number of collisions is greater than the augmentation. This can be easily expressed with a simple formula depending on the parameters.

This is the key point in our Membrane Computing algorithm, since we do not need to perform the movement and then to check is we have an improvement, we can evaluate it *a priori*, by exploring the objects placed in the skin. Obviously, if the squares share a descendant diagonal, the situation is analogous.

If the queens do not share diagonal, the study is analogous, but the obtained formula by considering that we get a *feasible movement* if the reduction is greater than the augmentation is slightly different.

From a technical point of view, we consider a finite set of rules with the following interpretation: *If the corresponding set of objects*

$u_{i-j}(UA_{i-j}, UB_{i-j}, UC_{i-j}, UD_{i-j})$	$d_{i+j}(DA_{i+j}, DB_{i+j}, DC_{i+j}, DD_{i+j})$
$u_{k-s}(UA_{k-s}, UB_{k-s}, UC_{k-s}, UD_{k-s})$	$d_{k+s}(DA_{k+s}, DB_{k+s}, DC_{k+s}, DD_{k+s})$
$u_{i-s}(UA_{i-s}, UB_{i-s}, UC_{i-s}, UD_{i-s})$	$d_{i+s}(DA_{i+s}, DB_{i+s}, DC_{i+s}, DD_{i+s})$
$u_{k-s}(UA_{k-s}, UB_{k-s}, UC_{k-s}, UD_{k-s})$	$d_{k+j}(DA_{k+j}, DB_{k+j}, DC_{k+j}, DD_{k+j})$

is placed in the skin, then the movement of queen from (i, j) *and* (k, s) *to* (i, s) *and* (k, j) *improves the number of collisions.*

In the general case, there will be many possible applications of rules of this type. The P system choose one of them in a non-deterministic way. The application of

the rule introduces an object $change_{ijks}$ in the skin. After a complex set of rules, this object produces a new configuration and the cycle starts again.

The design of the P system depends on N, the number of queens and it is full of technical details. It uses cooperation, inhibitors and electrical charges in order to control the flow of objects. In particular, a set of rules halts the P system if a board with zero collisions is reached and another set of rules re-starts the P system (produces a configuration equivalent to the initial one) if no more improvements can be achieved from the current configuration.

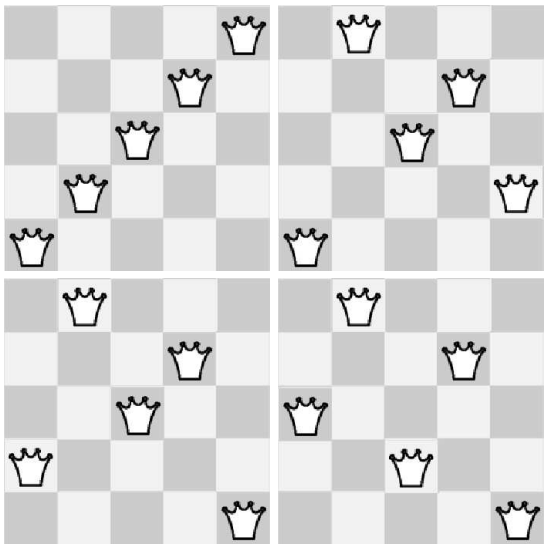


Fig. 2. Starting from a configuration C_0 with 4 collisions (up-left) we can reach C_1 with 3 collisions (up-right) and then C_2 with 2 collisions (bottom-left) and finally C_3 with 0 collisions (bottom-right), which is a solution to the 5-queens problem.

Figure 2 shows a solution found with the corresponding P system for the 5-queens problem. We start with a board with all the queens in the main ascendant diagonal (up-left). The collisions in this diagonal (and in the whole board) is 4. By changing the queens from the columns 2 and 5, we obtain the board shown in Figure 1 with 3 collisions (up-right in Figure 2). In the next step, the queens form columns 1 and 5 are changed, and we get a board with 2 collisions, produced because the two main diagonals have two queens each (bottom-left). Finally, by changing the queens is the columns 1 and 3, we get a board with no collisions that represent a solution to the 5-queens problem (bottom-right).

5 Experimental Results

An *ad hoc* CLIPS program was written *inspired* on this Membrane Computing design. Some experiments were performed on a system with an Intel Pentium Dual CPU E2200 at 2,20 GHz, 3GB of RAM and using CLIPS V6.241 under the operating system Windows Vista.

Due to the random choosing of the initial configuration and the non-determinism of the P system for choosing the movement, 20 experiments have been performed for each number of queens N for $N \in \{10, 20, \ldots, 200\}$ in order to get an informative parameter. We have considered the average of these 20 experiments on the number of P system steps and the number of seconds. The following table shows the result of the experiments.

Number of queens	Average of steps	Average of secs.
10	141.35	0.0171549
20	166.25	0.133275
30	270.9	0.717275
40	272.7	1.71325
50	382.4	4.75144
60	453.85	9.65071
70	495.45	16.9358
80	637.6	33.1815
90	625	47.3944
100	757.6	80.6878
110	745.75	113.635
120	841.75	157.937
130	891.25	216.141
140	983.7	311.71
150	979.75	381.414
160	1093	541.022
170	1145.5	683.763
180	1206.25	872.504
190	1272.256	1089.13
200	1365.25	1423.89

Notice, for example, that in the solution presented in [4], the solution for 20 queens was obtained after 15,944 seconds. The average time obtained with this approach is 0.133275 seconds.

6 Final Remarks

Due to the high computational cost of classical methods, local search has became an alternative for searching solution to real-life hard problems [1, 5].

In this paper we present a first approach to the problem of local search by using Membrane Computing and we have applied to the N-queens problem as a case study. As future work, several possibilities arise: One of them is to improve the design from a P system point of view, maybe considering new ingredients; a second one is to consider new case studies closer to real-life problems; a third one is to implement the design in parallel architectures and compare the results with the obtained ones with an one-processor computer.

Acknowledgements

The authors acknowledge the support of the projects TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence of the Junta de Andalucía, grant P08-TIC-04200.

References

1. Bijarbooneh, F.H., Flener, P., Pearson, J.: Dynamic demand-capacity balancing for air traffic management using constraint-based local search: First results. In: Deville, Y., Solnon, C. (eds.) LSCS. EPTCS, vol. 5, pp. 27–40 (2009)
2. 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, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5957, pp. 264–288. Springer (2009)
3. Gutiérrez-Naranjo, M.A., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Solving the N-queens puzzle with P systems. In: Gutiérrez-Escudero, R., Gutiérrez-Naranjo, M.A., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Seventh Brainstorming Week on Membrane Computing. vol. I, pp. 199–210. Fénix Editora, Sevilla, Spain (2009)
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Depth-first search with P systems. 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. 257–264. Springer (2010)
5. Hoos, H.H., Stützle, T.: Stochastic Local Search : Foundations & Applications (The Morgan Kaufmann Series in Artificial Intelligence). Morgan Kaufmann, 1 edn. (Sep 2004)
6. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–285 (2003)
7. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach (2nd Edition). Prentice Hall (December 2002)
8. Sosic, R., Gu, J.: Efficient local search with conflict minimization: A case study of the N-queens problem. IEEE Transactions on Knowledge and Data Engineering 6(5), 661–668 (1994)

Notes About Spiking Neural P Systems

Mihai Ionescu[1], Gheorghe Păun[2,3]

[1] University of Piteşti
Str. Târgu din Vale, nr. 1, 110040 Piteşti
Romania
mihaiarmand.ionescu@gmail.com
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
[3] Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Summary. Spiking neural P systems (SN P systems, for short) are much investigated in the last years in membrane computing, but still many open problems and research topics are open in this area. Here, we first recall two such problems (both related to neural biology) from [15]. One of them asks to build an SN P system able to store a number, and to provide it to a reader without losing it, so that the number is available for a further reading. We build here such a memory module and we discuss its extension to model/implement more general operations, specific to (simple) data bases. Then, we formulate another research issue, concerning pattern recognition in terms of SN P systems. In the context, we define a recent version of SN P systems, enlarged with rules able to request spikes from the environment; based on this version, so-called SN dP systems were recently introduced, extending to neural P systems the idea of a distributed dP automaton. Some details about such devices are also given, as a further invitation to the reader to this area of research.

1 Introduction

The present notes are only an invitation to the reader to a recent and vividly investigated branch of membrane computing, inspired from the way the neurons cooperate in large nets, communicating (among others) by means of spikes, electrical impulses of identical shapes. In neural computing, this biological reality has inspired a series of research which are considered "neural computing of the third generation", see, e.g., [5], [12]. In terms of membrane computing, the idea was captured in the form of so-called *spiking neural P systems*, in short, SN P systems, introduced in [10] and then investigated in a large number of papers. We refer to the Handbook [20] and to the membrane computing website [23] for details.

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. Such a rule is of the form $E/a^c \to a^p; d$, where E is a regular expression over the alphabet $\{a\}$ (a denotes the spike); such a rule can be used in a neuron if the number of spikes present in the neuron is described by the regular expression E (if there are k spikes in the neuron, then $a^k \in L(E)$, where $L(E)$ is the regular language identified by E), and using it means consuming c spikes (hence $k - c$ remain) and producing p spikes, which will be sent to all neurons to which a synapse exists which leaves the current neuron, after a time delay of d steps (if $d = 0$, then the spikes leave immediately). If a neuron can use a rule, then it has to use one, hence the system is synchronized, in each time unit, all neurons which can spike should do it. One starts from an initial configuration and one proceed by computation steps as suggested above. One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train*. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the string corresponding to the sequence of spikes which exit the output neuron. This sequence is a binary one, with 0 associated with a step when no spike is emitted and 1 associated with a step when a spike is emitted.

Several variants were considered in the literature. We recall here the most recent one, SN P systems with request rules, proposed in [9]: also rules of the form $E/\lambda \leftarrow a^r$ are used, with the meaning that r spikes are brought in the neuron, provided that its content is described by the regular expression E.

Such rules are essentially used in defining SN dP systems, a class of distributed computing devices bridging the SN P systems area and the dP systems area – this latter one initiated in [16] and then investigated in [4], [17], [18]. We recall here the definition of SN dP systems from [9], as well an example from that paper.

Also, two research topics mentioned in the last years (especially in the framework of the Brainstorming Week on Membrane Computing, organized at the beginning of each February in Sevilla, Spain – see [23]) are briefly discussed, for the first one also providing a preliminary answer (which, in turn, raises further questions). Namely, the challenge was to define a SN P module which simulate the "memory function" of the brain: stores a number, which can then be read by another "part of the brain" without losing the respective number. Such a module is provided here, but it suggests a series of continuations in terms of (simple) data bases, where several "memory cells" can be considered, loaded and interrogated, removed and added. Continuing our construction remains a task for the reader, and similarly with the second problem: to construct an SN P system able to recognize patterns, in a precise way which will be defined below.

In short, this is only a quick introduction to the study of SN P systems, by giving a few basic definitions and a short list of recent notions and research topics. The interested reader should look for further details in the domain literature.

2 Formal Language and Automata Theory Prerequisites

We need below only a few basic elements of automata and language theory, and of computability theory. It would be useful for the reader to have some familiarity with such notions, e.g., from [21] and [22], but, for the sake of readability we introduce here the notations and notions used later in the paper.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. If $x = a_1 a_2 \dots a_n$, $a_i \in V$, $1 \leq i \leq n$, then $mi(x) = a_n \dots a_2 a_1$.

We denote by REG, RE the families of regular and recursively enumerable languages. The family of Turing computable sets of numbers is denoted by NRE (these sets are length sets of RE languages, hence the notation).

In the rules of spiking neural P systems we use the notion of a *regular expression*; given an alphabet V, (i) λ and each $a \in V$ is a regular expression over V, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. The non-necessary parentheses can be omitted, while $E_1^+ \cup \lambda$ can be written as E_1^*. With each expression E we associate a language $L(E)$ as follows: (i) $L(\lambda) = \{\lambda\}, L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1)(E_2)) = L(E_1)L(E_2)$, $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, and $L((E_1)^+) = L(E_1)^+$, for any regular expressions E_1, E_2.

The operations used here are the standard union, concatenation, and Kleene +. We also need below the operation of the *right derivative* of a language $L \subseteq V^*$ with respect to a string $x \in V^*$, which is defined by

$$L/x = \{y \in V^* \mid yx \in L\}.$$

In the following sections, when comparing the power of two language generating/accepting devices the empty string λ is ignored.

3 Spiking Neural P Systems with Request Rules

We directly introduce the type of SN P systems we investigate in this paper; the reader can find details about the standard definition in [10], [19], [2], etc.

An *(extended) spiking neural P system* (abbreviated as *SN P system*) with *request rules*, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_0),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);

2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:
a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
b) R_i is a finite set of *rules* of the forms
   (i) $E/a^c \to a^p$, where E is a regular expression over a and $c \geq p \geq 1$ (spiking rules);
   (ii) $E/\lambda \leftarrow a^r$, where E is a regular expression over a and $r \geq 1$ (request rules);
   (iii) $a^s \to \lambda$, with $s \geq 1$ (forgetting rules) such that there is no rule $E/a^c \to a^p$ of type (i) or $E/\lambda \leftarrow a^r$ of type (ii) with $a^s \in L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (σ_{i_0}) of the system.

A rule $E/a^c \to a^p$ is applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule can *fire*, and its application means consuming (removing) c spikes (thus only $k - c$ remain in σ_i) and producing p spikes, which will exit immediately the neuron. A rule $E/\lambda \leftarrow a^r$ is used if the neuron contains k spikes and $a^k \in L(E)$; no spike is consumed, but r spikes are added to the spikes in σ_i. In turn, a rule $a^s \to \lambda$ is used if the neuron contains exactly s spikes, which are removed ("forgotten"). A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

If a rule $E/a^c \to a^p$ has $E = a^c$, then we will write it in the simplified form $a^c \to a^p$.

The spikes emitted by a neuron σ_i go to all neurons σ_j such that $(i, j) \in syn$, i.e., if σ_i has used a rule $E/a^c \to a^p$, then each neuron σ_j receives p spikes. The spikes produced by a rule $E/\lambda \leftarrow a^r$ are added to the spikes in the neuron and to those received from other neurons, hence they are counted/used in the next step of the computation.

If several rules can be used at the same time, then the one to be applied is chosen non-deterministically.

During the computation, a configuration of the system is described by the number of spikes present in each neuron; thus, the initial configuration is described by the numbers $n_1, n_2, \ldots, n_m$.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule can be used.

There are many possibilities to associate a result with a computation, in the form of a number (the distance between two input spikes or two output spikes) or of a string. Like in [3], we associate a symbol b_i with a step of a computation when i spikes exit the system, thus generating strings over an alphabet $\{b_0, b_1, \ldots, b_m\}$,

for some $m \geq 1$. When one neuron is distinguished as an input neuron, then the sequence of symbols b_i associated as above with the spikes taken from the environment by this input neuron also forms a string. In both cases, we can distinguish two possibilities: to interpret b_0 as a symbol or to simply ignore a step when no spike is read or sent out. The second case provides a considerable freedom, as the computation can proceed inside the system without influencing the result, and this adds power to our devices.

In what follows, we also consider an intermediate case: the system can work inside for at most a given number of steps, k, before reading or sending out a symbol. (Note the important detail that this is a *property* of the system, not a *condition* about the computations: all halting computations observe the restriction to work inside for at most k steps, this is not a way to select some computations as correct and to discard the others. This latter possibility is worth investigating, but we do not examine it here.) The obtained languages, in the accepting and the generating modes, are denoted $L_k^a(\Pi), L_k^g(\Pi)$, respectively, where $k \in \{0, 1, 2, \ldots\} \cup \{\infty\}$. Then, L_0^g corresponds to the restricted case of [3] and L_∞^g to the non-restricted case (denoted L_λ in [3]).

The respective families of languages associated with systems with at most m neurons are denoted by $L_k^\alpha SNP_m$, where $\alpha \in \{g, a\}$ and k is as above; if k is arbitrary, but not ∞, then we replace it with $*$; if m is arbitrary, then we replace it with $*$. (Note that we do not take here into account the descriptional complexity parameters usually considered in this framework: number of rules per neuron, numbers of spikes consumed or forgotten, etc.)

The computing power of the previous devices was preliminarily investigated in [9], with many questions still remaining open. We do not recall them here, but, instead, we mention the notion of a SN dP system introduced in [9].

4 SN dP Systems

We first recall from [9], without proofs, two basic results, because they provide a way to find counterexamples in this area. Actually, they are extensions to SN P systems with request rules of some results already proved in [3].

Lemma 1. *The number of configurations reachable after n steps by an extended SN P system with request rules of degree m is bounded by a polynomial $g(n)$ of degree m.*

Theorem 1. *If $f : V^+ \longrightarrow V^+$ is an injective function, $card(V) \geq 2$, then there is no extended SN P system Π with request rules such that $L_f(V) = \{x\, f(x) \mid x \in V^+\} = L_*^g(\Pi)$.*

Corollary 1. *The following two languages are not in $L_*^g SNP_*$ (in all cases, $card(V) = k \geq 2$):*

$$L_1 = \{x\, mi(x) \mid x \in V^+\},$$
$$L_2 = \{xx \mid x \in V^+\}.$$

Note that language L_1 above is a non-regular minimal linear one and L_2 is context-sensitive non-context-free.

We introduce now the mentioned distributed version of SN P systems:

An *SN dP system* is a construct

$$\Delta = (O, \Pi_1, \ldots, \Pi_n, esyn),$$

where (1) $O = \{a\}$ (as usual, a represents the spike), (2) $\Pi_i = (O, \sigma_{i,1}, \ldots, \sigma_{i,k_i}, syn, in_i)$ is an SN P system with request rules present only in neuron σ_{in_i} ($\sigma_{i,j} = (n_{i,j}, R_{i,j})$, where $n_{i,j}$ is the number of spikes initially present in the neuron and $R_{i,j}$ is the finite set of rules of the neuron, $1 \leq j \leq k_i$), and (3) $esyn$ is a set of *external synapses*, namely between neurons from different systems Π_i, with the restriction that between two systems Π_i, Π_j there exist at most one link from a neuron of Π_i to a neuron of Π_j and at most one link from a neuron of Π_j to a neuron of Π_i. We stress the fact that we allow request rules only in neurons σ_{in_i} of each system Π_i – although this restriction can be removed; the study of this extension remains as a task for the reader. The systems $\Pi_i, 1 \leq i \leq n$, are called *components* (or *modules*) of the system Δ.

As usual in dP automata, each component can take an input (by using request rules), work on it by using the spiking and forgetting rules in the neurons, and communicate with other components (along the synapses in $esyn$); the communication is done as usual inside the components: when a spiking rule produces a number of spikes, they are sent simultaneously to all neurons, inside the component or outside it, in other components, provided that a synapse (internal or external) exists to the destination.

As above, when r spikes are taken from the environment, a symbol b_r is associated with that step, hence the strings we consider introduced in the system are over an alphabet $V = \{b_0, b_1, \ldots, b_k\}$, with k being the maximum number of spikes introduced in a component by a request rule.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over V if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, working in the synchronous (in each time unit, each neuron which can use a rule should use one) non-deterministic way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

Hence, the SN dP systems are synchronized, a universal clock exists for all components and neurons, marking the time in the same way for the whole system.

In what follows, like in the communication complexity area, see, e.g., [8], we ask the components to take equal parts of the input string, modulo one symbol. (One also says that the string is distributed *in a balanced way*. The study of the unbalanced (free) case remains as a research issue.) Specifically, for an SN dP system Δ of degree n we define the language $L(\Delta)$, of all strings $x \in V^*$ such that we can write $x = x_1 x_2 \ldots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the computation halts. Moreover, we can distinguish between considering b_0 as a symbol or not, like in the previous sections, thus obtaining the languages $L_\alpha(\Delta)$, with $\alpha \in \{0, 1, 2, \ldots\} \cup \{\infty, *\}$.

Let us denote by $L_\alpha SNdP_n$ the family of languages $L_\alpha(\Delta)$, for Δ of degree at most n and $\alpha \in \{0,1,2,\ldots\} \cup \{\infty, *\}$. An SN dP system of degree 1 is a usual SN P system with request rules working in the accepting mode (with only one input neuron). Thus, the universality of SN dP systems is ensured, for the case of languages $L_\infty(\Delta)$.

In what follows, we prove the usefulness of distribution, in the form of SN dP systems, by proving that one of the languages in Corollary 1, can be recognized by a simple SN dP system (with two components), even working in the L_k mode.

Proposition 1. $\{ww \mid w \in \{b_1, b_2, \ldots, b_k\}^*\} \in L_{k+2}SNdP_2$.

Proof. The SN dP system which recognizes the language in the proposition is the following:

$$
\begin{aligned}
\Delta &= (\{a\}, \Pi_1, \Pi_2, \{((2,1),(1,3)),\ ((1,5),(2,1))\}),\ \text{with the components}\\
\Pi_1 &= (\{a\}, \sigma_{(1,1)}, \ldots, \sigma_{(1,7)}, syn_1, (1,1)),\\
&\quad \sigma_{(1,1)} = (3, \{a^3/\lambda \leftarrow a^r \mid 1 \le r \le k\} \cup \{a^4a^+/a \rightarrow a,\ a^4 \rightarrow a^3\}),\\
&\quad \sigma_{(1,2)} = (0, \{a \rightarrow a,\ a^3 \rightarrow a^3\}),\\
&\quad \sigma_{(1,3)} = (0, \{a \rightarrow a,\ a^3 \rightarrow a^3\}),\\
&\quad \sigma_{(1,4)} = (0, \{a^2 \rightarrow \lambda,\ a^6 \rightarrow \lambda,\ a \rightarrow a,\ a^4 \rightarrow a\}),\\
&\quad \sigma_{(1,5)} = (0, \{a^2 \rightarrow \lambda,\ a^6 \rightarrow a,\ a^6 \rightarrow a^3\}),\\
&\quad \sigma_{(1,6)} = (0, \{a^+/a \rightarrow a\}),\\
&\quad \sigma_{(1,7)} = (0, \{a^+/a \rightarrow a\}),\\
&\quad syn_1 = \{((1,1),(1,2)),\ ((1,5),(1,1)),\ ((1,2),(1,4)),\ ((1,4),(1,6)),\\
&\qquad\quad ((1,4),(1,7)),\ ((1,6),(1,7)),\ ((1,7),(1,6)),\ ((1,2),(1,5)),\\
&\qquad\quad ((1,3),(1,4)),\ ((1,3),(1,5))\},\\
\Pi_2 &= (\{a\}, \sigma_{(2,1)}, \emptyset, (2,1)),\\
&\quad \sigma_{(2,1)} = (3, \{a^3/\lambda \leftarrow a^r \mid 1 \le r \le k\} \cup \{a^4a^+/a \rightarrow a,\ a^4 \rightarrow a^3\}).
\end{aligned}
$$

The proof that this system works properly, recognizing indeed the language in the statement of the proposition, can be found in [9].

Many problems can be formulated for SN P systems with request rules and for SN dP systems. Several were formulated in [9], from where we recall the following two general ones. :esides the synchronized (sequential in each neuron) mode of evolution, there were also introduced other modes, such as the exhaustive one, [11], and the non-synchronized one, [1]. Universality was proved for these types of SN P systems, but only for the extended case. Can universality be proved for non-extended SN P systems also using request rules?

5 Two Research Topics About SN P Systems

Many research topics and open problems about SN P systems can be found in the literature. We mention here only the collection from [14], and we recall two of the problems formulated in [15].

G. Continuing with SN P systems, a problem which was vaguely formulate from time to time, but only orally, refers to a basic feature of the brain, the memory. How can this be captured in terms of SN P systems is an intriguing question. First, what means "memory"? In principle, the possibility to store some information for a certain time (remember that there is a short term and also a long term memory), and to use this information without losing it. For our systems, let us take the case of storing a number; we need a module of an SN P system where this number is "memorized" in such a way that in precise circumstances (e.g., at request, when a signal comes from outside the module), the number is "communicated" without "forgetting" it. In turn, the communication can be to only one destination or to several destinations. There are probably several ways to build such a module. The difficulty comes from the fact that if the number n is stored in the form of n spikes, "reading" these spikes would consume them, hence it is necessary to produce copies which in the end of the process reset the module. This is clearly possible in terms of SN P systems, what remains to do is to explicitly write the system. However, new questions appear related to the efficiency of the construction, in terms of time (after getting the request for the number n, how many steps are necessary in order to provide the information and to reset the module?), and also in terms of descriptional complexity (how many neurons and rules, how many spikes, how complex rules?). It is possible that a sort of "orthogonal" pair of ideas are useful: many spikes in a few neurons (n spikes in one neuron already is a way to store the number, what remains is to read and reset), or a few spikes in many neurons (a cycle of n neurons among which a single spike circulates, completing the cycle in n steps, is another "memory cell" which stores the number n; again, we need to read and reset, if possible, using only a few spikes). Another possible question is to build a reusable module, able to store several numbers: for a while (e.g., until a special signal) a number n_1 is stored, after that another number n_2, and so on.

H. The previous problem can be placed in a more general set-up, that of modeling other neurobiological issues in terms of SN P systems. A contribution in this respect is already included in the present volume, [13], where the sleep-awake passage is considered. Of course, the approach is somewhat metaphorical, as the distance between the physiological functioning of the brain and the formal structure and functioning of an SN P system is obvious, but still this illustrates the versatility and modeling power of SN P systems. Further biological details should be considered in order to have a model with some significance for the brain study (computer simulations will then be necessary, like in the case of other applications of P systems in modeling biological processes). However, also at this formal level there are several problems to consider. For instance, what happens if the sleeping

period is shortened, e.g., because a signal comes from the environment? Can this lead to a "damage" of the system? In general, what about taking the environment into account? For instance, we can consider a larger system, where some modules sleep while other modules not; during the awake period it is natural to assume that the modules interact, but not when one of them is sleeping, excepting the case of an "emergency", when a sleeping module can be awakened at the request of a neighboring module. Several similar scenarios can be imagined, maybe also coupling the sleep-awake issue with the memory issue.

The first of these problems will be answered below.

6 A Memory Module

We start by directly given the memory module which answers the requests of problem **G** from [15]; we present it in a graphical form, in Figure 1, using the standard way of representing SN P systems (neurons as nodes of a graph, linked by arrows which represent synapses, with the rules of each neuron written in the respective nodes, together with the spikes initially present there; input and output neurons have incoming or outgoing arrows, respectively).

The system in Figure 1 works as follows. The number n is introduced in the memory module, in the form of a spike train containing n occurrences of 1, one after the other. The computation starts when the first spike enters the system – at that moment, rule $a^5/a^3 \to a$ is enabled. For the other spikes, the rule $a^3/a \to a$ is used, always two spikes remaining inside neuron σ_1. If no spike enters the system, then the two existing spikes are removed. If, at a subsequent step, any spike enters neuron σ_1, then it is forgotten, too, by means of the rule $a \to \lambda$. Thus, the number to be stored should be introduced as a compact sequence of spikes.

Each input spike is doubled by neurons σ_2, σ_3, and in this way $2n$ spikes are accumulated in neuron σ_4. They can stay here forever, unchanged. If the trigger neuron, σ_9, receives a spike (we assume that this happens after completing the introduction of the n spikes in neuron σ_1), then a further spike is sent to neuron σ_4. With an odd number of spikes inside, neuron σ_4 consumes two by two the spikes, moving in this way n spikes in the "beneficiary" neuron σ_{10}, at the same time moving $2n$ spikes to neuron σ_8. Note that after exhausting the spikes of neuron σ_4, one further spike is produced by σ_7, hence in the end neuron σ_8 gets an odd number of spikes. In the same way as σ_4 has moved its contents to σ_8, now the reverse operation takes place, hence σ_4 will end with $2n$ spikes inside (and σ_8 is empty). Therefore, the "cell memory" is restored, the number n can be read again, when the trigger gets one further spike. Always, the number n is made available outside the memory module, but the module "remembers" the number, for a further usage.

Now, may extensions can be imagined, for instance, towards data bases. A table in a data base can be imagined as a sequence of "memory cells" (modules), each one with its own label and value (number stored) and subject to updating

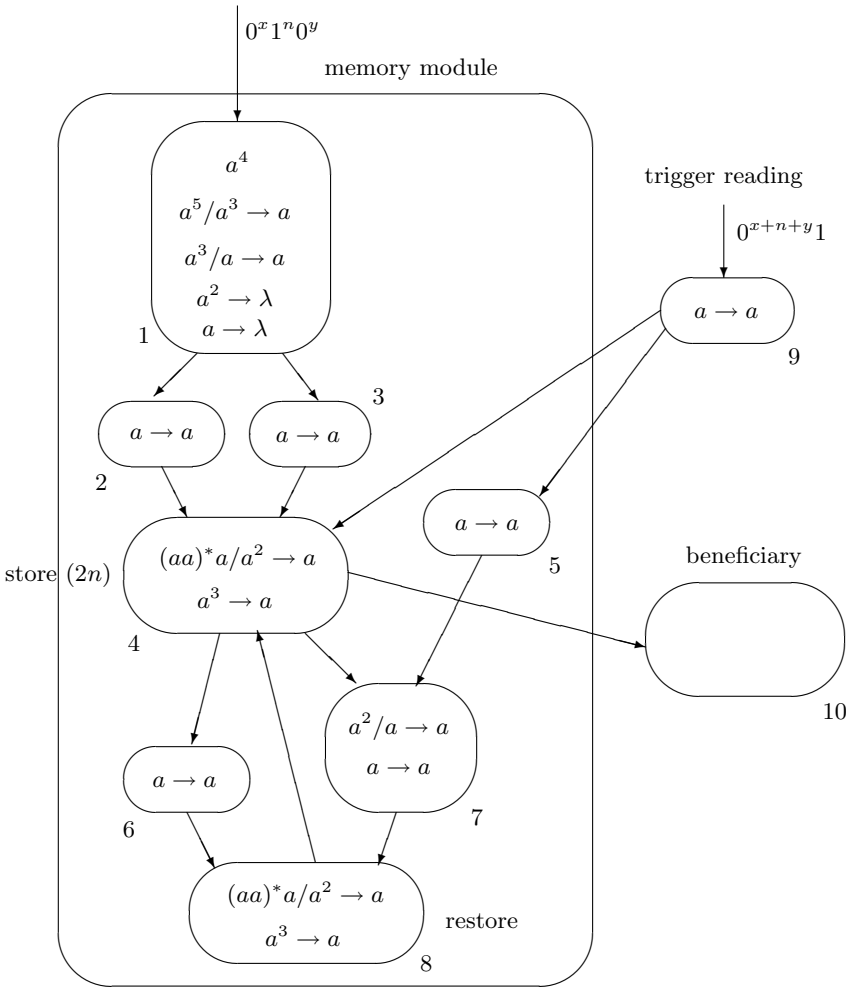


Fig. 1. A memory module

and interrogating operations. Modules can also be inserted and removed. Each of these operations are done by means of a specialized module, one for updating, one for interrogating, and so on, always specifying first the "address" (the label, the ID) of the cell which is operated. The construction becomes complex (only the indicated modules should be modified, although the trigger for the respective operation can do the same with all other modules), but presumably feasible; we leave this task to the reader.

Another direction of research is to consider other "brain modules" or functions and implement them in terms of SN P systems. An example is that from [13], where

the sleeping activity of the brain is modeled, but many others can be addressed: learning, getting tired, taking the task of another part of the brain.

7 Pattern Recognition

One function of the brain which we have not mentioned above is pattern recognition. Neural computing is especially concerned with this task, and learning (training the net) is an essential step of pattern recognition. Although the problem is so natural, only a few efforts were paid in SN P systems area to incorporating the learning feature, see, e.g., [7]. In turn, nothing was done in what concerns the patterns recognition, that it why we call here the attention about this problem.

A simple framework for that concerns handling – generating or recognizing – arrays, pictures realized by marking the positions of a grid with symbols of a given alphabet, in the sense of array grammars and languages. For instance, we can imagine the following task. Consider a language of arrays, for instance, the letters of the Latin alphabet, each one written in a rectangle with black and white lattice positions. We can interpret a marked spot (black) as a spike and a non-marked one (white) as no spike. Let us construct an SN P system having certain input neurons aligned, able to read one by one the rows of the array picture. The reading can be done in consecutive steps or internal computations are allowed between reading two neighboring rows. After reading the whole array, the system can continue working and, if the computation halts, the input array is recognized.

The difficulty of constructing such an SN P system lies in the fact that we have to recognize several different letters. A variant is to construct a system which recognizes only one of the letters, say A, but of different sized. In this latter case, the number of input neurons should be large enough for reading all possible sizes of the letter, and the difficulty lies again in the fact that the same neurons can behave differently for letters of different sizes.

Anyway, we find the construction of such patter recognition SN P systems interesting, non-trivial and rather instructive, a good exercise for understanding the functioning of SN P systems, and, hopefully, a way to find applications for them.

8 Final Remarks

We insist about the fact that this is only an invitation to a dynamical research area of membrane computing, trying to convince the reader that there are interesting problems to address and providing a few bibliographical hints. For comprehensive presentations and references, the reader should consult the domain literature, available at [23] (for instance, all brainstorming volumes can be found there, in a downloadable form).

Acknowledgements

The work of M. Ionescu was possible due to CNCSIS grant RP-4 12/01.07.2009. The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Science*, 410, 24-25 (2009), 2352–2364.
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, 1-4 (2007), 141–162
3. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.
4. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest University. Mathematics-Informatics Series*, 63 (2009), 5–22.
5. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
6. R. Gutierrez-Escudero et al.: *Proceedings of the Seventh Brainstorming Week on Membrane Computing*. Sevilla, 2009, 2 volume, Fenix Editora, Sevilla, 2009.
7. M.A. Gutierrez-Naranjo, M.J. Pérez-Jiménez: A first model for Hebbian learning with spiking neural P systems. In *Proc. 6th Brainstorming Week on Membrane Computing*, Sevilla, 2008.
8. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing*. Springer, Berlin, 1997.
9. M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez T. Yokomori: Spiking neural dP systems. In *Proc. 9th Brainstorming Week on Membrane Computing*, Sevilla, January 31– February 4, 2011.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
11. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.
12. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, 1999.
13. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. In [6], vol. 2, 59–72.
14. Gh. Păun: Twenty six research topics about spiking neural P systems. In *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, Fenix Editora, Sevilla, 2007, 263–280
15. Gh. Păun: Some open problems collected during 7th BWMC. In [6], vol. 2, 197–206.
16. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.

17. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Lecture Notes in Computer Science*, 6570, in press.
18. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Sci.*, in press.
19. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
20. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
21. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
22. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
23. The P Systems Website: `http://ppage.psystems.eu`.

Spiking Neural P Systems with Several Types of Spikes

Mihai Ionescu[1], Gheorghe Păun[2,3],
Mario J. Pérez-Jiménez[3], Alfonso Rodríguez-Patón[4]

[1] University of Piteşti
Str. Târgu din Vale, nr. 1, 110040 Piteşti, Romania
armandmihai.ionescu@gmail.com
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
[3] Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es
[4] Department of Artificial Intelligence, Faculty of Computer Science
Polytechnical University of Madrid, Campus de Montegancedo
Boadilla del Monte 28660, Madrid, Spain
arpaton@fi.upm.es

Summary. With a motivation related to gene expression, where enzymes act in series, somewhat similar to the train spikes traveling along the axons of neurons, we consider an extension of spiking neural P systems, where several types of "spikes" are allowed. The power of the obtained spiking neural P systems is investigated and the modeling of gene expression in these terms is discussed. Some further extensions are mentioned, such as considering a process of decay in time of the spikes.

1 Introduction

The present note lies at the intersection of two active research branches of bioinformatics/natural computing, namely, gene expression and membrane computing. Specifically, an extension of so-called spiking neural P systems (in short, SN P systems) is considered, with motivations related to gene expression processes.

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. Such a rule is has the general form $E/a^c \to a^p; d$, where E is a regular expression (equivalently, we can consider it a regular language) and a denotes the spike; if the contents of the neuron is described by an element of the regular language (identified by)

E, then the rule is enabled, c spikes are consumed and p are produced, and sent, after a time delay of d steps, along the synapses leaving the neuron. There also are forgetting rules of the form $a^c \rightarrow \lambda$, with the meaning that, if the neuron contains exactly c spikes, then they can be removed (forgotten). One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train*. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the string corresponding to the sequence of spikes which exit the output neuron.

These computing devices were introduced in [7] and then investigated in a large number of papers; we refer to the corresponding chapter from [13] and to the membrane computing website [16] for details.

In turn, gene expression is an important research area where various transcription factors appears and, important for their activity, their frequency matters – see, for instance, [1], [10], [12]. This means that a spiking like process is encountered, but with several "spikes", the regulator proteins which bind to a promotor depending on their concentration. In some sense we have here a communication process in which a signal encoded in a concentration (the transcription factor) is transduced to a frequency signal (the bursts of mRNA associated to the bindings of the transcription factor with the promotor) and again transduced back to a concentration (the level of protein produced). Thus, conceptually, we can approach this process in terms of theoretical machineries developed for spiking neurons – with the necessity of considering a variety of spikes, not only one as in the neural case. This is also suggested in [1]: "...we anticipate that frequency-modulated regulation may represent a general principle by which cells coordinate their response to signals."

Starting from these observations, we relate here the two research areas, introducing SN P systems with several types of spikes. Such a possibility was somehow forecasted already from the way the definition in [7] is given, with an alphabet, O, for the set of spikes, but with only one symbol in O; up to now, only a second type of spikes was considered, in [11], namely *anti-spikes*, which, when introduced, are immediately annihilated, in pairs with usual spikes. This extension to several types of spikes is natural also in view of the fact that all classes of P systems investigated in membrane computing work with arbitrary alphabets of objects.

As expected, having several types of spikes helps in proofs; in particular, we obtain the universality of the SN P systems with several types of spikes for systems with a very reduced number of neurons – remember that for systems with only one type of spikes the proofs do not bound the number of neurons (but such a bound can be found due to the existence of universal SN P systems, hence with a fixed number of neurons, but used in the computing mode, having both an input and an output). Three ways to define the result of a computation are considered: as the number of objects inside a specified neuron, as the number of objects sent

out by the output neuron, and as the distance in time between the first two spikes sent out during the computation.

What is not investigated is the case of generating strings, in the sense of [3], [4], or even in the distributed case of [8]. Other open problems are mentioned in the rest of the paper and in the final section of it.

2 Formal Language Theory Prerequisites

We assume the reader to be familiar with basic language and automata theory, e.g., from [14] and [15], so that we introduce here only some notations and notions used later in the paper.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. For a language L, we denote by $sub(L)$ the set of all substrings of strings in L.

As usual in membrane computing, the multisets over a finite universe set U are represented by strings in U^* (two strings equal modulo a permutation represent the same multiset). If $u, v \in U^*$, we write the fact that u is a submultiset of v in the form $u \subseteq v$, with the understanding that there is a permutation of v having u as a substring (this can be formally formulated also in terms of Parikh mapping, but we do not enter into details). Similarly, we write $u \in sub(L)$ for a multiset u and a set L of multisets, meaning that u is a submultiset of a multiset in L.

A register machine (in the non-deterministic version) is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction `HALT`), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n present in register 1 at that time is said to be generated by M. Without loss of generality we may assume that in the halting configuration all other registers are empty. It is known that register machines generate all sets of numbers which are Turing computable – we denote this family

with NRE (RE stands for "recursively enumerable"). By $NFIN$ we denote the family of finite sets of natural numbers.

In the following sections, when comparing the power of two computing devices, number 0 is ignored (this corresponds to the fact that when comparing the power of language generating or accepting devices, the empty string λ is ignored).

3 Spiking Neural P Systems with Several Types of Spikes

We directly introduce the type of SN P systems we investigate in this paper; although somewhat far from the idea of a spike from the neural area, we still call the objects processed in our devices *spikes*.

A *spiking neural P system with several types of spikes* (abbreviated as *SN*$^+$ *P system*, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0), \text{ where:}$$

1. O is the alphabet of *spikes* (we also say *objects*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (w_i, R_i), 1 \leq i \leq m$, where:
   a) $w_i \in O^*$ is the *initial multiset* of spikes contained in σ_i;
   b) R_i is a finite set of *rules* of the forms
      (i) $E/u \rightarrow a$, where E is a regular language over O, $u \in O^+$, and $a \in O$ (spiking rules);
      (ii) $v \rightarrow \lambda$, with $v \in O^+$ (forgetting rules) such that there is no rule $E/u \rightarrow a$ of type (i) with $v \in E$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (σ_{i_0}) of the system.

A rule $E/u \rightarrow a$ is applied as follows. If the neuron σ_i contains a multiset w of spikes such that $w \in L(E)$ and $u \in sub(w)$, then the rule can *fire*, and its application means consuming (removing) the spikes identified by u and producing the spike a, which will exit immediately the neuron. In turn, a rule $v \rightarrow \lambda$ is used if the neuron contains exactly the spikes identified by v, which are removed ("forgotten"). A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

If a rule $E/u \rightarrow a$ has $E = \{u\}$, then we will write it in the simplified form $u \rightarrow a$.

The spike emitted by a neuron σ_i go to all neurons σ_j such that $(i, j) \in syn$.

If several rules can be used at the same time in a neuron, then the one to be applied is chosen non-deterministically.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule can be used.

There are many possibilities to associate a result with a computation, in the form of a number. Three possibilities are considered here: the number of objects in the output neuron in the halting configuration, the number of spikes sent to the environment by the output neuron, and the number of steps elapsed between the first two steps when the output neuron spikes. In the first two cases only halting computations provide an output, in the last case we can define the output also for ever going computations – but in what follows we only work with halting computations also for this case.

We denote by $N_\alpha(\Pi)$ the set of numbers generated as above by an SN^+ P system Π with the result defined in the mode $\alpha \in \{i, o, d\}$, where i indicate the internal output, o the external one (as the number of spikes), and d the fact that we count the distance between the first two spikes which exit the system. Then, $N_\alpha SN^+P_m$ is the family of sets of numbers $N_\alpha(\Pi)$, for SN^+ P systems with at most $m \geq 1$ neurons. As usual, the subscript m is replaced by $*$ if the number of neurons does not matter.

Before passing to investigate the size of the previously defined families, let us mention that we have introduced here SN P systems of the *standard* type in what concerns the rules, i.e., producing only one spike, and without *delay*; *extended* rules are natural ($E/u \to v$, with both u and v multisets), but this is a too general case from a computability point of view, corresponding to cooperating P systems. It is important also to note that the rules we use have both additional powerful features – context sensitivity induced by the existence of the control regular language E, and strong restrictions – the produced spike (only one) should leave immediately the neuron, it cannot be further used in the same place without being sent back by the neighboring neurons. These features are essentially present in the proofs from the next section.

4 The Power of SN^+ P Systems

We start by considering the case when the result is counted inside the system (like in general P systems, hence somewhat far from the style of SN P systems).

Lemma 1. $NRE \subseteq N_iSN^+P_3$.

Proof. Let us consider a register machine $M = (n, H, l_0, l_h, I)$. We construct the following SN^+ P system

$$
\begin{aligned}
\Pi &= (O, \sigma_1, \sigma_2, \sigma_3, syn, 1), \text{ with:}\\
O &= \{a_i \mid 1 \leq i \leq n\} \cup \{l, l', l'' \mid l \in H\},\\
\sigma_1 &= (l_0, R_1),\\
R_1 &= \{O^*/l_i \to l'_i \mid (l_i : \mathtt{ADD}(r), l_j, l_k) \in I\}\\
&\quad \cup \{O^*a^rO^*/l_ia_r \to l''_j,\ (O^* - O^*a^rO^*)/l_i \to l''_k \mid (l_i : \mathtt{SUB}(r), l_j, l_k) \in I\}\\
&\quad \cup \{l_h \to \lambda\},
\end{aligned}
$$

$$\sigma_2 = (\lambda, R_2),$$
$$R_2 = \{l_i' \to l_j,\ l_i' \to l_k \mid (l_i : \texttt{ADD}(r), l_j, l_k) \in I\} \cup \{l'' \to l \mid l \in H\},$$
$$\sigma_3 = (\lambda, R_3),$$
$$R_3 = \{l_i' \to a_r \mid (l_i : \texttt{ADD}(r), l_j, l_k) \in I\} \cup \{l'' \to \lambda \mid l \in H\},$$
$$syn = \{(1,2),\ (1,3),\ (2,1),\ (3,1)\}.$$

The functioning of this system can be easily followed. The contents of register r is represented by the number of copies of object a_r present in the system. There are also objects associated with the labels of M.

Initially, we have only the object l_0 in neuron σ_1. In general, in the presence of a label l_i of an instruction in I, the instruction is simulated by the system Π. For the ADD instructions, the change of labels is done with the help of neuron σ_2 and the addition of a further object a_r is done in neuron σ_3. For the SUB instructions, the check for zero is performed by means of the regular language associated with the rules in R_1. The computation continues as long as the work of the machine M continues. When the label l_h is introduced – by the neuron σ_2 – the computation stops after one further step, when this object is removed from the output neuron, σ_1. Thus, in the end, this neuron only contains copies of object a_1, hence their number represents the value present in the first register of M in the end of the computation. Thus, $N(M) = L_i(\Pi)$. □

Theorem 1. $NFIN = N_iSN^+P_1 = N_iSN^+P_2 \subset N_iSN^+P_3 = NRE$.

Proof. The inclusions $N_iSN^+P_1 \subseteq N_iSN^+P_2 \subseteq N_iSN^+P_3$ are obvious from the definitions. The inclusion $N_iSN^+P_3 \subseteq NRE$ is straightforward (we can also invoke for it the Turing-Church thesis).

In an SN^+ P system with two components, the number of spikes present inside the two neurons cannot be increased (each spiking rule consumes at least one spike and produces only one spike, while there is no duplication of spikes because of multiple synapses which exit a neuron), hence we have $N_iSN^+P_2 \subseteq NFIN$.

On the other hand, $NFIN \subseteq N_iSN^+P_1$. Indeed, consider a finite set of numbers, $F = \{n_1, n_2, \ldots, n_k\}$; assume that $1 \leq n_1 < n_2 < \ldots < n_k$ (remember that we ignore the number 0). We construct the system

$$\Pi = (\{a\}, (a^{n_k+1}, \{a^{n_k+1}/a^{n_k+1-n_i} \to a \mid 1 \leq i \leq k\}), \emptyset, 1).$$

We have $L_i(\Pi) = F$: each computation has only one step, which nondeterministically uses one of the rules in R_1. Each such rule just consumes a number of spikes, passing from the initial $n_k + 1$ spikes to any number $n_i \in F$, which cannot be further processed.

Together with Lemma 1, this concludes the proof of the theorem. □

Let us note in the construction from the proof of Lemma 1 that all neurons spike a large number of times (related to the length of the computation), not directly related to the number computed in the first register of M. This makes

difficult to imagine a system with only three neurons which is universal when the result is defined as the number of spikes sent out. However, one additional neuron suffices in such a case.

Theorem 2. $NFIN = N_oSN^+P_1 \subset N_oSN^+P_2 \subseteq N_oSN^+P_3 \subseteq N_oSN^+P_4 = NRE$.

Proof. Again, the inclusions $N_oSN^+P_1 \subseteq N_oSN^+P_2 \subseteq N_oSN^+P_3 \subseteq N_oSN^+P_4 \subseteq NRE$ are obvious from the definitions.

The inclusion $NRE \subseteq N_oSN^+P_4$ can be obtained by a slight extension of the construction in the proof of Lemma 1: we replace the rule $l_h \to \lambda$ from R_1 with the rule

$$a_1^+ l_h / l_h a_1 \to l_h.$$

We also add a neuron σ_4, considered as output neuron, linked by synapses (1, 4), (4, 1) to the neuron σ_1 and containing the unique rule

$$l_h \to l_h.$$

When the computation of M stops, hence l_h is introduced in σ_1, this object remove one by one the objects a_1 and moves to the output neuron. This neuron both sends l_h out and back to σ_1, hence the number of copies of l_h sent out is equal with the number stored in the first register of M.

This time, an SN^+ P system with two components can compute an arbitrarily large number, by sending out an arbitrarily large number of spikes. For instance,

$$\Pi = (\{a\}, (a, \{a \to a\}), (aa, \{aa/a \to a,\ aa \to a\}), \{(1,2),(2,1)\}, 2),$$

has $L_o(\Pi) = \{1, 2, \ldots\}$ (neuron σ_2 spikes step by step, until using the rule $aa \to$, when only one spike remains in the system and the computation halts).

If we have only one neuron, the computation can last as may steps as many spikes are initially inside, hence $N_oSN^+P_1 \subseteq NFIN$.

On the other hand, $NFIN \subseteq N_iSN^+P_1$. Indeed, consider again a finite set of numbers, $F = \{n_1, n_2, \ldots, n_k\}$ such that $1 \leq n_1 < n_2 < \ldots < n_k$ and construct the system

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \emptyset, 1), \text{ with} \\
\sigma_1 &= (a^{n_k+1}, R_1), \\
R_1 &= \{a^{n_k+1}/a^{n_k+1-n_i+1} \to a \mid 1 \leq i \leq k\} \\
&\quad \cup \{a^r/a \to a \mid 1 \leq r \leq n_k - 1\}.
\end{aligned}$$

We have $L_i(\Pi) = F$: each computation starts with a step which uses non-deterministically a rule $a^{n_k+1}/a^{n_k+1-n_i+1} \to a$, which decreases the number of spikes from the initial $n_k + 1$ to some $n_i - 1$; at this time, one spike was sent out. From now on, we use deterministically rules of the form $a^r/a \to a$, for all $r = 1, 2, \ldots, n_i - 1$, hence for $n_i - 1$ steps, always sending out one spike. Thus, in total, we send out n_i spikes, for each $n_i \in F$.

Combining all these remarks, we have the theorem. □

It is an *open problem* whether or not the inclusions $N_oSN^+P_2 \subseteq N_oSN^+P_3 \subseteq N_oSN^+P_4$ are proper.

Theorem 3. $NFIN = N_dSN^+P_1 = N_dSN^+P_2 \subset N_dSN^+P_3 \subseteq N_dSN^+P_4 = NRE$.

Proof. As above, the inclusions $N_dSN^+P_1 \subseteq N_dSN^+P_2 \subseteq N_dSN^+P_3 \subseteq N_dSN^+P_4 \subseteq NRE$ are obvious from the definitions.

The inclusion $NFIN \subseteq N_dSN^+P_1$ is already proved for SN P systems with only one type of spikes. Like in that case, we also obtain the inclusion $N_dSN^+P_2 \subseteq NFIN$: in order to generate an arbitrarily large number, the output neuron should not spike for an arbitrarily large number of steps, but this is not possible in a system with only two neurons, because if only one neuron is working, it can perform only a number of steps bounded by the number of spikes initially present in it.

The fact that $N_dSN^+P_3$ contains infinite sets of numbers is also known for standard SN P systems.

What remains to prove is the inclusion $NRE \subseteq N_dSN^+P_4$ and this can again be obtained by an extension of the construction in the proof of Lemma 1; because this extension is not immediate, we give the construction in full details.

We consider a register machine $M = (n, H, l_0, l_h, I)$ and construct the SN^+ P system Π as indicated in Figure 1 – this time we do not give the system formally, but we represent it graphically, in the way usual in the SN P systems area.

The work of this system is identical to that in the proof of Lemma 1, until producing the object l_h (the objects which arrive in the output neuron σ_4 from all other neurons remain here unused).

When σ_2 introduces the object l_h, it is sent to all other neurons. It waits unused in σ_4, but in σ_1 and σ_3 it is reproduced in each step, hence these two neurons feed repeatedly each other with one copy of l_h. In σ_1, each use of the rule $a_1^+ l_h / a_1 l_h \to l_h$ removes one copy of a_1. In the end of step 1 (we count here only the steps after having l_h in the system, hence for the phase when the output is produced), neuron σ_4 contains three copies of l_h. Thus, in step 2, this neuron spikes.

From now on, neurons σ_1, σ_3 spike repeatedly, exchanging copies of l_h, σ_4 always forgets the two copies of l_h received from σ_1, σ_3 (while σ_2 just accumulates copies of l_h, which cannot be processed here). When the last copy of a_1 is removed from σ_1 (if m copies of a_1 were present here when l_h was introduced, then this happens in step m, after having l_h in the system), this is the last step when σ_4 receives two spikes. In the next step $(m+1)$ it receives only the spike produced by σ_3, which is used (in step $m+2$) by the rule $l_h \to l_h$ in σ_4. The computation stops. The number of steps between the two spikes sent out by the output neuron is $(m+2) - 2 = m$, hence the number computed by the register machine in its first register.

The proof of the theorem is now complete. □

It is an *open problem* whether or not the inclusion $N_gSN^+P_3 \subseteq N_gSN^+P_4$ is proper.

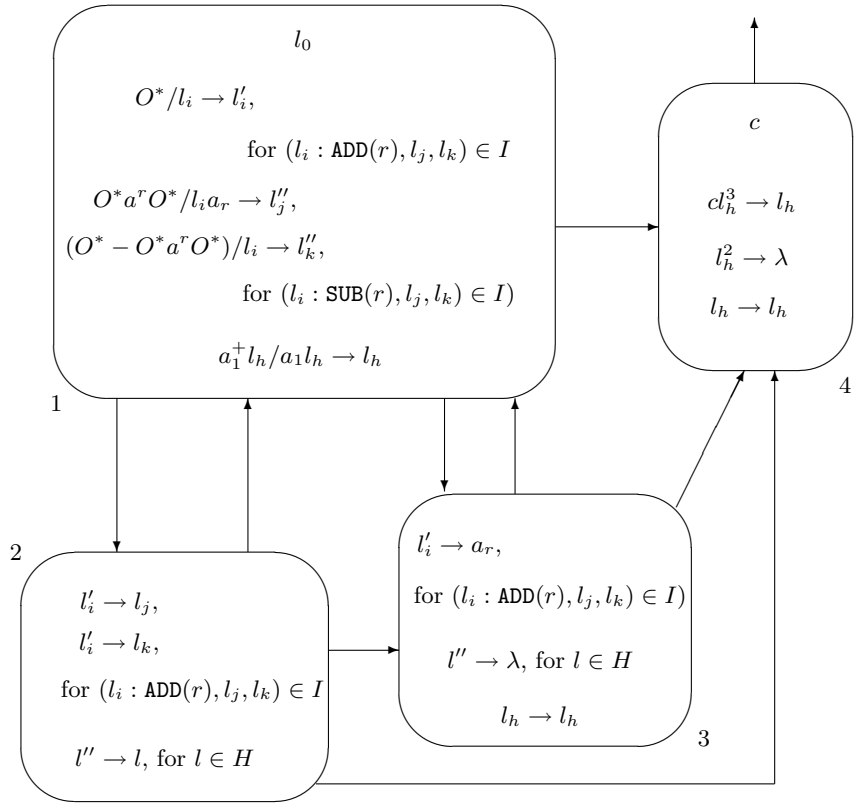


Fig. 1. The SN^+ P system from the proof of Theorem 3

5 Final Remarks

In gene expression it is also the case that the enzymes have a time dependency of their reactivity, which can be captured in terms of SN P systems by considering decaying spikes, in the sense of [6]. For instance, we can associate an *age* with each produced spike, by using rules of the form $E/u \to (a, t)$, where $t \geq 1$ is the "duration of life" of this spike. If the spike is not used in a step, then its life is decreased by one unit (this is like having rewriting rules $(a, s) \to (a, s-1)$, used in parallel for all spikes not used in spiking or forgetting rules), until reaching the state $(a, 0)$, when a rule $(a, 0) \to \lambda$ is assumed to be applied. This feature remains to be further investigated.

Let us close by recalling the fact that besides the synchronized (sequential in each neuron) mode of evolution, there were also introduced other modes, such as the exhaustive one, [9], and the non-synchronized one, [2], which also deserve to be considered for SN P systems with several types of spikes.

Acknowledgements

The work of M. Ionescu was possible due to CNCSIS grant RP-4 12/01.07.2009. The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. L. Cai, C.K. Dalal, M.B. Elowitz: Frequency-modulated nuclear localization bursts coordinate gene regulation. *Nature*, 455 (25 September 2008).
2. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Science*, 410, 24-25 (2009), 2352–2364.
3. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162.
4. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.
5. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.
6. R. Freund, M. Ionescu, M. Oswald: Extended spiking neural P systems with decaying spikes and/or total spiking. *Intern. J. Found. Computer Sci.*, 19 (2008), 1223–1234.
7. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
8. M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: T. Yokomori: Spiking neural dP systems. *Proc. Ninth Brainstorming Week on Membrane Computing*, Sevilla, 2011, RGNC Report 01/2011.
9. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.
10. E.M. Ozbudak, M. Thattai, I. Kurtser, A.D. Grossman, A. van Oudenaarden: Regulation of noise in the expression of a single gene. *Nature Genetics*, 31 (May 2002).
11. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. *Intern. J. Computers, Comm. Control*, 4, 3 (2009), 273–282.
12. J. Paulsson: Models of stochastic gene expression. *Physics of Life Reviews*, 2 (2005), 157-175.
13. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
14. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
15. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
16. The P Systems Website: `http://ppage.psystems.eu`.

Spiking Neural dP Systems

Mihai Ionescu[1], Gheorghe Păun[2,3],
Mario J. Pérez-Jiménez[3], Takashi Yokomori[4]

[1] University of Piteşti
Str. Târgu din Vale, nr. 1, 110040 Piteşti
Romania
armandmihai.ionescu@gmail.com
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
[3] Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es
[4] Department of Mathematics, School of Education
Waseda University, 1-6-1 Nishi-waseda, Shinjuku-ku
Tokyo 169-8050, Japan
yokomori@waseda.jp

Summary. We bring together two topics recently introduced in membrane computing, the much investigated spiking neural P systems (in short, SN P systems), inspired from the way the neurons communicate through spikes, and the dP systems (distributed P systems, with components which "read" strings from the environment and then cooperate in accepting their concatenation). The goal is to introduce SN dP systems, and to this aim we first introduce SN P systems with the possibility to input, at their request, spikes from the environment; this is done by so-called *request rules*. A preliminary investigation of the obtained SN dP systems (they can also be called *automata*) is carried out. As expected, request rules are useful, while the distribution in terms of dP systems can handle languages which cannot be generated by usual SN P systems. We always work with extended SN P systems; the non-extended case, as well as several other natural questions remain open.

1 Introduction

We combine here two ideas recently considered in membrane computing, the spiking neural P systems (in short, SN P systems) introduced in [9], and the dP systems introduced in [13].

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes)

along synapses (edges of the graph), under the control of firing rules. One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train*. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the string corresponding to the sequence of spikes which exit the output neuron. This sequence is a binary one, with 0 associated with a step when no spike is emitted and 1 associated with a step when a spike is emitted.

Actually, we use *extended* SN P systems, that is, we allow rules of the form $E/a^c \to a^p$, with the following meaning: if the content of the neuron is described by the regular expression E, then c spikes are consumed and p are produced and sent to the neurons to which there exist synapses leaving the neuron where the rule is applied (more precise definitions will be given in Section 3). In this way, strings over arbitrary alphabets can be obtained: if i spikes are sent out at the same time, then we say that the symbol b_i was generated. The languages generated by SN P systems in this way were investigated in [2] and [3]; see also [4].

In turn, a dP systems consists of several "modules" (usual P systems), which communicate among them by means of antiport rules like in tissue P systems. When only symport/antiport rules are used inside modules, then we can define a language accepted by such a system: each component takes from the environment sequences of symbols, they work separately and communicate through antiport rules and, if the computation halts, then the concatenation of the input strings is accepted. See [6], [13], [15], [16] for a series of results about such machineries; in particular, [6] investigates the language families characterized by P and dP automata, their relationships and place in Chomsky hierarchy.

There is an apparent difference between the two classes of P systems: SN P systems *generate* strings, while dP systems *accept* them. There were considered SN P systems also working in the accepting mode, with a spike train (a number is encoded as the distance between two consecutive spikes) introduced in a specified neuron of the system in the first steps of the computation. However, a more natural way to proceed, more similar to the way the P automata take symbols from the environment, is to consider a new type of rules in SN P systems, able to take spikes from the environment. We consider such rules of a form rather similar to spiking rules, namely, of the form $E/\lambda \leftarrow a^r$: if the contents of the neuron is described by the regular expression E, then r spikes are brought from the environment. Such a rule can be applied as a usual spiking rule (its use lasts one time unit, with the spikes brought from the environment being added to the contents of the neuron and ready to be used in the next step).

Now, the definition of an SN P system with request rules is rather natural: the neurons which contain request rules are supposed as having a "synapse" also with the environment such that they can take spikes from the environment, depending on the number of spikes they contain. The computation proceeds as usual, starting from the initial configuration. Both input and output spike trains can be associated with the neurons linked with the environment. Also the step to SN dP systems

(we can call them *automata*) is natural: take "modules" (or components) consisting of neurons, with only one neuron of each component also having a synapse with the environment (hence able to take an input); each component can be linked with another component through synapses among their neurons (for simplicity, we consider only the case when at most one synapse is available in each direction). The components take strings from the environment and the concatenation of these strings is accepted if the computation of the system halts.

Many questions arise in this framework. Compare usual SN P systems with SN P systems having request rules, both in the generative and the accepting mode. Do the additional facilities provided by the request rules help, e.g., from the point of view of the descriptional complexity? What about imposing a bound on the environment (considering that in the beginning it only contains a given number of spikes, and further spikes can be there only if the system sends spikes out)? More interesting: which is the power of SN dP systems? As expected, it is larger than that of SN P systems (with the mentioning that we compare languages *accepted* by SN dP systems with languages *generated* by SN P systems).

In this context, we introduce a refinement in the way of defining languages associated with (extended) SN P systems. In between the restricted (in each step, one symbol is produced) and the non-restricted case we can consider the languages generated/accepted in the k-restricted way: between reading or producing two symbols which are considered in the string, the system can work at most k steps without reading or sending symbols out.

2 Formal Language and Automata Theory Prerequisites

We assume the reader to be familiar with basic language and automata theory, e.g., from [18] and [19], so that we introduce here only some notations and notions used later in the paper.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. If $x = a_1 a_2 \dots a_n$, $a_i \in V$, $1 \leq i \leq n$, then $mi(x) = a_n \dots a_2 a_1$.

We denote by REG, RE the families of regular and recursively enumerable languages. The family of Turing computable sets of numbers is denoted by NRE (these sets are length sets of RE languages, hence the notation).

In most universality proofs in membrane computing, in particular, in the SN P systems area, one uses register machines (several registers can contain natural numbers; they are increased by one or decreased by one – the latter operation only after checking whether the number stored in the register is different from zero –, by means of labeled instructions; if the computation starting in an initial label with all registers empty halts by reaching a special halting label, then the number stored in the halting configuration by the first register is said to be computed/generated by that computation.

Because our SN P systems can read at the same time several spikes, an operation which corresponds to reading a symbol from an alphabet with several elements, we will use in the universality proof a device more adequate to this case: *counter automata with an input tape*. The version we choose is one with as simple as possible instructions: check for zero, increment, decrement (by one in both cases), read a symbol from the input tape, halt. Formally, such a device is a construct $M = (n, V, H, l_0, l_h, I)$, where n is the number of registers/counters, V is the alphabet of the input tape, H is the set of labels, each one uniquely associated with an instruction, l_0 is the initial label, l_h is the halt label, and I is the set of instructions of the following forms:

1. $(l_i : \texttt{check}(r), l_j, l_k)$: when label l_i is reached, the contents of register r is compared to zero; if the register is empty, then the next label is l_j, if the contents of the register is strictly positive, then the next label is l_k;
2. $(l_i : \texttt{add}(r), l_j)$: add 1 to the contents of register r and pass from label l_i to label l_j;
3. $(l_i : \texttt{sub}(r), l_j)$: subtract 1 from the contents of register r and pass from label l_i to label l_j (the operation is supposed to be possible, meaning that previously the register was checked for zero by an instruction of the first type);
4. $(l_i : \texttt{read}(b), l_j)$: read the symbols $b \in V$ from the tape and pass from label l_i to label l_j;
5. $l_h : \texttt{halt}$: when reaching l_h, the computation halts; this is the only instruction labeled by l_h.

Without loss of the generality, we may assume that in all instructions the involved labels are mutually different (this can be easily achieved by introducing intermediate labels, involved in additional instructions performing "dummy" operations, of the form "add 1 to register r, then subtract 1 from register r".

We start with label l_0 (by applying the instruction with label l_0), with all counters empty (storing the number 0), with the "reading head" in front of the first symbol of the input tape, where a string is written. We proceed as specified by the instructions, under the control of labels. The process is deterministic and the only branching is based on the check for zero of the registers. When the label l_h (hence the instruction $l_h : \texttt{halt}$) is reached, the computation stops and the sequence of symbols read from the input tape is the string accepted by the counter machine. The language of all such strings is denoted by $L(M)$.

It is known that counter machines (with a small number of counters, but this is not of interest below) with an input tape can recognize all recursively enumerable languages. Details (variants and proofs) can be found in several places: [11], [5], [7].

In the following sections, when comparing the power of two language generating/accepting devices the empty string λ is ignored.

3 Spiking Neural P Systems with Request Rules

We directly introduce the type of SN P systems we investigate in this paper; the reader can find details about the standard definition in [9], [14], [2], etc.

An *(extended) spiking neural P system* (abbreviated as *SN P system*) with *request rules*, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

where:

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

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

   where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the forms
      (i) $E/a^c \rightarrow a^p$, where E is a regular expression over a and $c \geq p \geq 1$ (spiking rules);
      (ii) $E/\lambda \leftarrow a^r$, where E is a regular expression over a and $r \geq 1$ (request rules);
      (iii) $a^s \rightarrow \lambda$, with $s \geq 1$ (forgetting rules) such that there is no rule $E/a^c \rightarrow a^p$ of type (i) or $E/\lambda \leftarrow a^r$ of type (ii) with $a^s \in L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (σ_{i_0}) of the system.

A rule $E/a^c \rightarrow a^p$ is applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule can *fire*, and its application means consuming (removing) c spikes (thus only $k - c$ remain in σ_i) and producing p spikes, which will exit immediately the neuron. A rule $E/\lambda \leftarrow a^r$ is used if the neuron contains k spikes and $a^k \in L(E)$; no spike is consumed, but r spikes are added to the spikes in σ_i. In turn, a rule $a^s \rightarrow \lambda$ is used if the neuron contains exactly s spikes, which are removed ("forgotten"). A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

If a rule $E/a^c \rightarrow a^p$ has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a^p$.

The spikes emitted by a neuron σ_i go to all neurons σ_j such that $(i, j) \in syn$, i.e., if σ_i has used a rule $E/a^c \rightarrow a^p$, then each neuron σ_j receives p spikes. The spikes produced by a rule $E/\lambda \leftarrow a^r$ are added to the spikes in the neuron and to those received from other neurons, hence they are counted/used in the next step of the computation.

If several rules can be used at the same time, then the one to be applied is chosen non-deterministically.

During the computation, a configuration of the system is described by the number of spikes present in each neuron; thus, the initial configuration is described by the numbers $n_1, n_2, \ldots, n_m$.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule can be used.

There are many possibilities to associate a result with a computation, in the form of a number (the distance between two input spikes or two output spikes) or of a string. Like in [3], we associate a symbol b_i with a step of a computation when i spikes exit the system, thus generating strings over an alphabet $\{b_0, b_1, \ldots, b_m\}$, for some $m \geq 1$. When one neuron is distinguished as an input neuron, then the sequence of symbols b_i associated as above with the spikes taken from the environment by this input neuron also forms a string. In both cases, we can distinguish two possibilities: to interpret b_0 as a symbol or to simply ignore a step when no spike is read or sent out. The second case provides a considerable freedom, as the computation can proceed inside the system without influencing the result, and this adds power to our devices.

In what follows, we also consider an intermediate case: the system can work inside for at most a given number of steps, k, before reading or sending out a symbol. (Note the important detail that this is a *property* of the system, not a *condition* about the computations: all halting computations observe the restriction to work inside for at most k steps, this is not a way to select some computations as correct and to discard the others. This latter possibility is worth investigating, but we do not examine it here.) The obtained languages, in the accepting and the generating modes, are denoted $L_k^a(\Pi), L_k^g(\Pi)$, respectively, where $k \in \{0, 1, 2, \ldots\} \cup \{\infty\}$. Then, L_0^g corresponds to the restricted case of [3] and L_∞^g to the non-restricted case (denoted L_λ in [3]).

The respective families of languages associated with systems with at most m neurons are denoted by $L_k^\alpha SNP_m$, where $\alpha \in \{g, a\}$ and k is as above; if k is arbitrary, but not ∞, then we replace it with $*$; if m is arbitrary, then we replace it with $*$. (Note that we do not take here into account the descriptional complexity parameters usually considered in this framework: number of rules per neuron, numbers of spikes consumed or forgotten, etc.)

By the definitions, we have the following inclusions for all $\beta \in \{1, 2, \ldots\} \cup \{*\}$:

$$L_0^\alpha SNP_\beta \subseteq L_1^\alpha SNP_\beta \subseteq L_2^\alpha SNP_\beta \subseteq \ldots L_*^\alpha SNP_\beta \subseteq L_\infty^\alpha SNP_\beta \subseteq RE.$$

4 Some Preliminary Results for the Generating Case

In general, the results which were obtained for SN P systems without request rules are expected to hold – maybe with simplified proofs – also for SN P systems with request rules. However, some differences exist. For instance, it is observed in

[2] that the language $\{0,1\}$ cannot be generated by an SN P system (the output neuron cannot choose between spiking or not spiking in the first step), but this language can be generated by the system

$$(\{a\}, (1, \{a \to a,\ a/\lambda \leftarrow a\}), \emptyset, 1),$$

because of the possibility of choosing between spiking or bringing a spike inside (the system halts after the first step). However, this is mainly due to the definition – allowing a nondeterministic choice between spiking and forgetting rules will lead to a similar result also for SN P systems without request rules.

Returning to the extension of results to the new class of SN P systems, we consider here three results from [3], as they are significant below.

Lemma 1. *The number of configurations reachable after n steps by an extended SN P system with request rules of degree m is bounded by a polynomial $g(n)$ of degree m.*

Proof. The same as in [3], with the observation that the number of spikes in the system is increased in two cases: when a neuron spikes, hence it introduces a well defined number of spikes in all neurons with which it has synapses, or when it brings spikes from the environment; in this case, the spikes, again a well defined number, are introduced in the neuron which has used the request rule. The rest of the argument remains the same as in [3]. □

Theorem 1. *If $f : V^+ \longrightarrow V^+$ is an injective function, $card(V) \geq 2$, then there is no extended SN P system Π with request rules such that $L_f(V) = \{x\, f(x) \mid x \in V^+\} = L^g_*(\Pi)$.*

Proof. Assume that there is an extended SN P system Π of degree m, with request rules, such that $L^g_k(\Pi) = L_f(V)$ for some f and V as in the statement of the theorem and some $k \geq 1$. According to the previous lemma, there are only polynomially many configurations of Π which can be reached after n steps. Take some n of the form $n = km$. The string generated after n steps is of length at least m. However, there are $card(V)^m \geq 2^m = 2^{n/k}$ strings of length m in V^+. Therefore, for large enough m there are two strings $w_1, w_2 \in V^+, w_1 \neq w_2$, such that after n steps the system Π reaches the same configuration when generating the strings $w_1\, f(w_1)$ and $w_2\, f(w_2)$, hence after step n the system can continue any of the two computations. This means that also the strings $w_1\, f(w_2)$ and $w_2\, f(w_1)$ are in $L^g_k(\Pi)$. Due to the injectivity of f and the definition of $L_f(V)$ such strings are not in $L_f(V)$, hence the equality $L_f(V) = L^g_k(\Pi)$ is contradictory. □

Corollary 1. *The following two languages are not in $L^g_* SNP_*$ (in all cases, $card(V) = k \geq 2$):*

$$L_1 = \{x\, mi(x) \mid x \in V^+\},$$
$$L_2 = \{xx \mid x \in V^+\}.$$

Note that language L_1 above is a non-regular minimal linear one and L_2 is context-sensitive non-context-free.

Theorems 3 and 4 from [3] (characterizing regular languages, modulo a symbol added to their strings), and Theorem 5 (generating non-semilinear languages), always in the restricted mode, can now be written in the form:

Theorem 2. $L^g_\infty SNP_2 \subseteq REG \subset L^g_1 SNP_3$.

5 The Accepting Case

In order to have a definition of the language accepted by a given SN P system with request rules, we have to distinguish a neuron as the input one, and only the sequence of symbols taken from the environment by that neuron is introduced in the language. The other neurons are allowed to bring spikes from the environment, but those spikes are not defining symbols for the processed string.

First, let us notice that Lemma 1 and Theorem 1 remains true also in the accepting case, hence the languages in Corollary 1 cannot be recognized.

Also the characterization of RE from [3] remains true. However, this proof is rather complex: one takes symbols from the environment, in the form of packages of spikes, but for a string w over an alphabet with k symbols, one passes to $val_{k+1}(w)$, the numerical value of w when considered as a number written in base $k+1$, and one accepts w if and only if $val_{k+1}(w)$ is accepted (a language L is in RE if and only if $val_{k+1}(L)$ is in NRE). Then, handling numbers is reduced to simulating register machines (basically, counter machines without an input tape). Here we will proceed in a direct way, proving that SN P systems with request rules can recognize all RE languages by starting from counter automata with input tape.

Theorem 3. $L^a_\infty SNP_* = RE$.

Proof. Let us consider a counter automaton $M = (n, V, H, l_o, l_h, I)$ as introduced in Section 2, with $V = \{b_1, \ldots, b_k\}$. We construct modules simulating instruction of the first four types mentioned in Section 2 – no halting module is necessary, we just ignore the halting instruction. Let us denote by Π the SN P system we construct.

For each counter r of M, Π contains a neuron $\sigma_r, 1 \leq r \leq n$; if the value of the counter will be at some moment m, then the associated neuron will contain $2m$ spikes. For each label $l \in H$, we also introduce in Π a neuron σ_l. All neurons of the system are empty in the beginning of the computation, except neuron σ_{l_0}, which contains one spike. Having a spike inside, this neuron is *active*; in general, when a neuron $\sigma_l, l \in H$, receives one spike, then it is active, the associated module will start working, simulating the instruction identified by the label l. There is a unique input neuron, with the label in, and σ_{in} is the only neuron of Π which contains request rules. Several other auxiliary neurons are involved in the modules. They and the synapses among all these neurons are specified below.

We do not give formally the modules, but in a graphical form.

Let us start with a CHECK instruction, $(l_i : \texttt{check}(r), l_j, l_k)$. We construct the module shown in Figure 1.

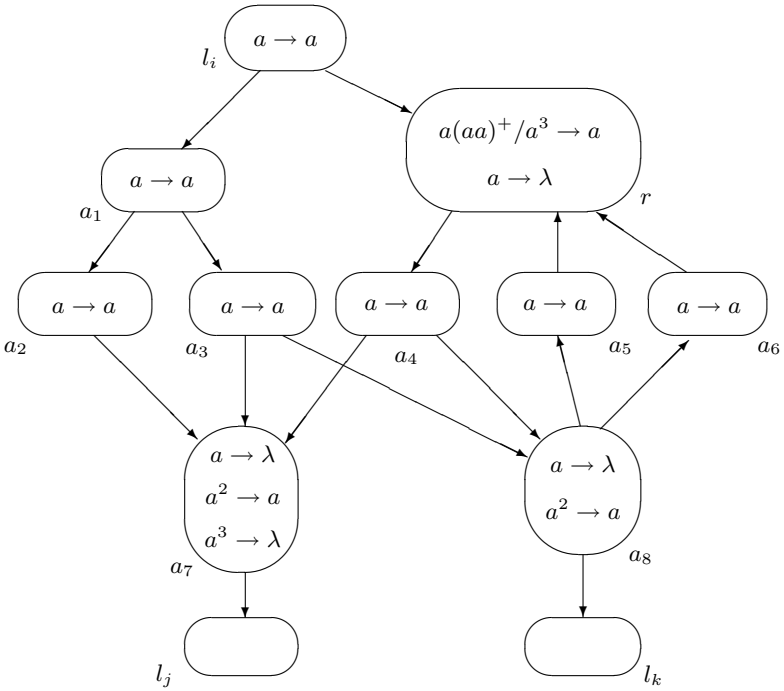


Fig. 1. The CHECK module

As long as the contents of a neuron σ_r is an even number, no rule can be applied in it. If σ_{l_i} becomes active, it sends a spike to σ_r, hence the number of spikes becomes odd. If this number is 1, hence the counter was empty, then a spike will eventually arrive in σ_{l_j}. If the counter was non-zero, then only σ_{a_8} will spike. In this way, it activates the neuron σ_{l_k} and also restores the contents of counter r, putting back the two spikes consumed by the rule $a(aa)^+/a^3 \to a$. The continuation is correct in both cases.

The new neurons, $\sigma_{a_s}, 1 \leq s \leq 8$, are uniquely associated with this module (it would be more rigorous to label them, say, by $(l_i, s), 1 \leq s \leq 8$, but we prefer the simple writing). All the three label neurons in Figure 1 can have incoming synapses from various other neurons, but each such neuron has only one associated module which is triggered by it. These remarks are valid for all modules constructed below.

Because several CHECK instructions (also several SUB instructions – see below) can act on the same counter r, it means that σ_r can have synapses to several

neurons of type σ_{a_4}, in various modules. However, this entails nothing wrong, because the spike produced by σ_{a_4} will be erased in both neurons $\sigma_{a_7}, \sigma_{a_8}$.

The modules for the ADD and SUB instructions are rather simple – they are given in Figure 2 (for $(l_i : \texttt{add}(r), l_j)$) and Figure 3 (for $(l_i : \texttt{sub}(r), l_j)$; remember that, before activating a SUB instruction, we assume that we have checked whether the operation can be done, that is, whether the counter is non-zero, hence we can assume that always the operation asked for by the instruction $(l_i : \texttt{sub}(r), l_j)$ is possible).

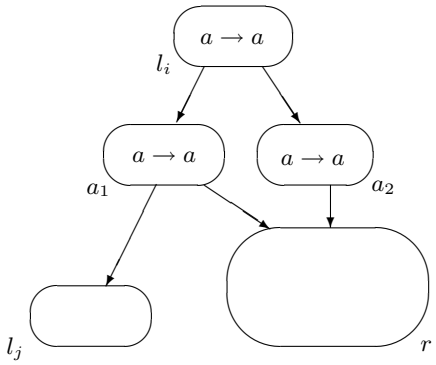


Fig. 2. The ADD module

In the SUB case, we have reproduced both rules of σ_r used in the CHECK module, as the first one is also used in the SUB case (but the produced spike is "lost", as explained before when discussing the CHECK module); the second rule, $a \to \lambda$, cannot be used, as we know in advance that the substraction is possible, hence the neuron is not empty.

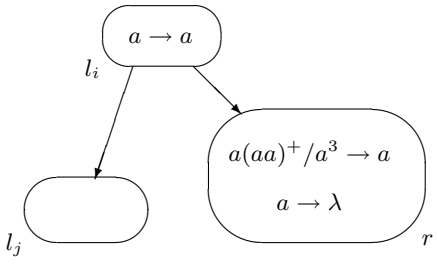


Fig. 3. The SUB module

Only the READ instructions $(l_i : \texttt{read}(b_s), l_j), 1 \leq s \leq k$, remains to be considered. For such an instruction, we build a module like in Figure 4. Note that the module contains several neurons, depending on s, and that the input neuron, labeled with in, is unique for the whole system.

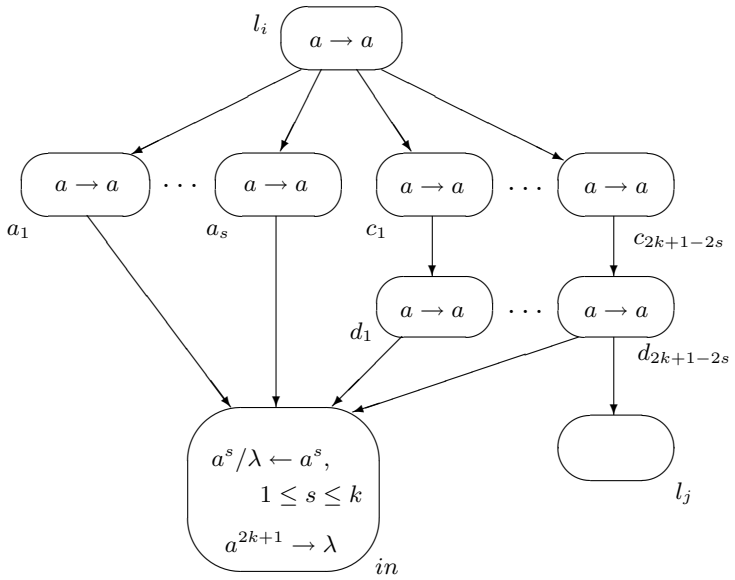


Fig. 4. The READ module

After activating σ_{l_i}, s spikes reach σ_{in}, and in this way the input neuron can take from the environment the correct number of spikes (reading in this way the symbol b_s). Because the input neuron can be used in any further step, it should be left empty after simulating the instruction $(l_i : \texttt{read}(b_s), l_j)$, and to this aim the use of the forgetting rule $a^{2k+1} \to \lambda$ is made possible, after receiving from neurons $\sigma_{d_t}, 1 \leq t \leq 2k+1-2s$, the corresponding number of spikes. One of these neurons, the last one, also sends a spike to σ_{l_j}, in order to continue the computation as requested by the READ instruction.

By repeatedly using these modules, the computation in M is correctly simulated; as pointed out above, there is no misleading interaction between modules. When the label l_h is reached in M, the unique neuron associated with a label which has a spike inside is σ_{l_h}, where we provide no rule, hence the computation halts also in Π. Consequently, the recognized string is the same, $L(M) = L^a_\infty(\Pi)$, and this completes the proof. □

Note the interesting fact that in the previous construction we only use non-extended spiking rules, always producing only one spike. If we allow the use of

extended rules, then some of the constructions can be slightly simplified (this is the case, for instance, with the modules CHECK).

6 SN dP Systems

We pass now to the main goal of our paper, introducing the SN P systems counterpart of dP systems from [13]. We directly introduce the definition of the systems we investigate.

An *SN dP system* is a construct

$$\Delta = (O, \Pi_1, \ldots, \Pi_n, esyn),$$

where (1) $O = \{a\}$ (as usual, a represents the spike), (2) $\Pi_i = (O, \sigma_{i,1}, \ldots, \sigma_{i,k_i}, syn, in_i)$ is an SN P system with request rules present only in neuron σ_{in_i} ($\sigma_{i,j} = (n_{i,j}, R_{i,j})$, where $n_{i,j}$ is the number of spikes initially present in the neuron and $R_{i,j}$ is the finite set of rules of the neuron, $1 \leq j \leq k_i$), and (3) $esyn$ is a set of *external synapses*, namely between neurons from different systems Π_i, with the restriction that between two systems Π_i, Π_j there exist at most one link from a neuron of Π_i to a neuron of Π_j and at most one link from a neuron of Π_j to a neuron of Π_i. We stress the fact that we allow request rules only in neurons σ_{in_i} of each system Π_i – although this restriction can be removed; the study of this extension remains as a task for the reader. The systems $\Pi_i, 1 \leq i \leq n$, are called *components* (or *modules*) of the system Δ.

As usual in dP automata, each component can take an input (by using request rules), work on it by using the spiking and forgetting rules in the neurons, and communicate with other components (along the synapses in $esyn$); the communication is done as usual inside the components: when a spiking rule produces a number of spikes, they are sent simultaneously to all neurons, inside the component or outside it, in other components, provided that a synapse (internal or external) exists to the destination.

As above, when r spikes are taken from the environment, a symbol b_r is associated with that step, hence the strings we consider introduced in the system are over an alphabet $V = \{b_0, b_1, \ldots, b_k\}$, with k being the maximum number of spikes introduced in a component by a request rule.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over V if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, working in the synchronous (in each time unit, each neuron which can use a rule should use one) non-deterministic way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

Hence, the SN dP systems are synchronized, a universal clock exists for all components and neurons, marking the time in the same way for the whole system.

In what follows, like in the communication complexity area, see, e.g., [8], we ask the components to take equal parts of the input string, modulo one symbol. (One

also says that the string is distributed *in a balanced way*. The study of the unbalanced (free) case remains as a research issue.) Specifically, for an SN dP system Δ of degree n we define the language $L(\Delta)$, of all strings $x \in V^*$ such that we can write $x = x_1 x_2 \dots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the computation halts. Moreover, we can distinguish between considering b_0 as a symbol or not, like in the previous sections, thus obtaining the languages $L_\alpha(\Delta)$, with $\alpha \in \{0, 1, 2, \dots\} \cup \{\infty, *\}$.

Let us denote by $L_\alpha SNdP_n$ the family of languages $L_\alpha(\Delta)$, for Δ of degree at most n and $\alpha \in \{0, 1, 2, \dots\} \cup \{\infty, *\}$. An SN dP system of degree 1 is a usual SN P system with request rules working in the accepting mode (with only one input neuron), as considered in Section 5. Thus, the universality of SN dP systems is ensured, for the case of languages $L_\infty(\Delta)$.

In what follows, we prove the usefulness of distribution, in the form of SN dP systems, by proving that one of the languages in Corollary 1, can be recognized by a simple SN dP system (with two components), even working in the L_k mode.

Proposition 1. $\{ww \mid w \in \{b_1, b_2, \dots, b_k\}^*\} \in L_{k+2} SNdP_2$.

Proof. The SN dP system which recognizes the language in the proposition is the following:

$$
\begin{aligned}
\Delta &= (\{a\}, \Pi_1, \Pi_2, \{((2,1),(1,3)),\ ((1,5),(2,1))\}),\ \text{with the components}\\
\Pi_1 &= (\{a\}, \sigma_{(1,1)}, \dots, \sigma_{(1,7)}, syn_1, (1,1)),\\
&\quad \sigma_{(1,1)} = (3, \{a^3/\lambda \leftarrow a^r \mid 1 \leq r \leq k\} \cup \{a^4 a^+/a \rightarrow a,\ a^4 \rightarrow a^3\}),\\
&\quad \sigma_{(1,2)} = (0, \{a \rightarrow a,\ a^3 \rightarrow a^3\}),\\
&\quad \sigma_{(1,3)} = (0, \{a \rightarrow a,\ a^3 \rightarrow a^3\}),\\
&\quad \sigma_{(1,4)} = (0, \{a^2 \rightarrow \lambda,\ a^6 \rightarrow \lambda,\ a \rightarrow a,\ a^4 \rightarrow a\}),\\
&\quad \sigma_{(1,5)} = (0, \{a^2 \rightarrow \lambda,\ a^6 \rightarrow a,\ a^6 \rightarrow a^3\}),\\
&\quad \sigma_{(1,6)} = (0, \{a^+/a \rightarrow a\}),\\
&\quad \sigma_{(1,7)} = (0, \{a^+/a \rightarrow a\}),\\
&\quad syn_1 = \{((1,1),(1,2)),\ ((1,5),(1,1)),\ ((1,2),(1,4)),\ ((1,4),(1,6)),\\
&\qquad\quad ((1,4),(1,7)),\ ((1,6),(1,7)),\ ((1,7),(1,6)),\ ((1,2),(1,5)),\\
&\qquad\quad ((1,3),(1,4)),\ ((1,3),(1,5))\},\\
\Pi_2 &= (\{a\}, \sigma_{(2,1)}, \emptyset, (2,1)),\\
&\quad \sigma_{(2,1)} = (3, \{a^3/\lambda \leftarrow a^r \mid 1 \leq r \leq k\} \cup \{a^4 a^+/a \rightarrow a,\ a^4 \rightarrow a^3\}).
\end{aligned}
$$

For an easier reference, the system is also presented graphically, in Figure 5.

Assume that we are in a configuration where both neurons $\sigma_{(1,1)}$ and $\sigma_{(2,1)}$ contain three spikes, as in the beginning of the computation. Each neuron can bring spikes inside, by using the rules $a^3/\lambda \leftarrow a^r$; if they bring the same number of spikes, then the system will return to a configuration as the one we started with, hence the computation can continue, otherwise the system will never halt.

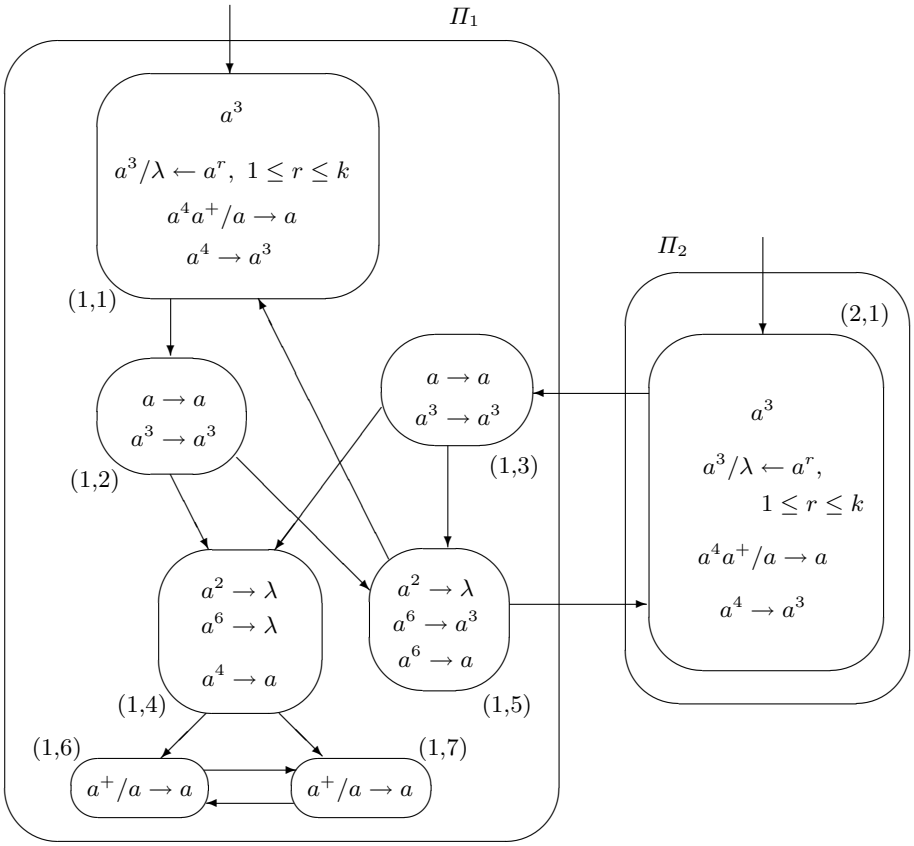


Fig. 5. The SN dP system from the proof of Proposition 1

For instance, assume that $\sigma_{(1,1)}$ brings inside r_1 spikes and $\sigma_{(2,1)}$ brings r_2 spikes. These spikes are moved one by one (at most k steps in total, where k is the cardinality of the alphabet) to neurons $\sigma_{(1,2)}$ and $\sigma_{(1,3)}$, respectively, by using the rules $a^4a^+/a \rightarrow a$, and from here, duplicated, in neurons $\sigma_{(1,4)}, \sigma_{(1,5)}$, where they are removed by the forgetting rules $a^2 \rightarrow \lambda$. If $r_1 = r_2$, then the neurons $\sigma_{(1,1)}, \sigma_{(2,1)}$ use at the same time the rules $a^4 \rightarrow a^3$, and after one further step six spikes reach both $\sigma_{(1,4)}$ and $\sigma_{(1,5)}$; in the former neuron the spikes are forgotten, in the latter one can use the rule $a^6 \rightarrow a^3$, which will send three spikes to $\sigma_{(1,1)}, \sigma_{(2,1)}$. The process can continue, one reads one further symbol of the string. If $r_1 \neq r_2$, then one of $\sigma_{(1,1)}, \sigma_{(2,1)}$ produces one spike and the other three, hence four spikes arrive in neuron $\sigma_{(1,4)}$; this neuron sends a spike to $\sigma_{(1,6)}$ and $\sigma_{(1,7)}$, and these neurons will exchange forever spikes, hence the computation never halts.

If, instead of the rule $a^6 \to a^3$, neuron $\sigma_{(1,5)}$ uses the rule $a^6 \to a$, then the computation stops, because the input neurons cannot fire having inside only one spike. This is the only way to stop the computation, hence the strings read by the two components are equal.

Note that after introducing some r spikes in each component of the system, we need $r-1$ steps for using the rules $a^4a^+/a \to a$, one step for the rule $a^4 \to a^3$ (at most k steps in total), then two more steps for sending three spikes (one in the end) to neurons $\sigma_{(1,1)}$ and $\sigma_{(2,1)}$. Therefore, $\{ww \mid w \in \{b_1, b_2, \ldots, b_k\}^*\} \in L_{k+2}SNdP_2$, and the proposition is proved. □

7 Final Remarks

Many problems can be formulated for SN P systems with request rules and for SN dP systems. Several were already mentioned in the previous sections. Let us close by recalling the fact that besides the synchronized (sequential in each neuron) mode of evolution, there were also introduced other modes, such as the exhaustive one, [10], and the non-synchronized one, [1]. Universality was proved for these types of SN P systems, but only for the extended case. Can universality be proved for non-extended SN P systems also using request rules?

Acknowledgements

The work of M. Ionescu was possible due to CNCSIS grant RP-4 12/01.07.2009. The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Science*, 410, 24-25 (2009), 2352–2364.
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, 1-4 (2007), 141–162.
3. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.
4. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.
5. P.C. Fischer: Turing machines with restricted memory access. *Information and Control*, 9 (1966), 364–379.

6. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest University. Mathematics-Informatics Series*, 63 (2009), 5–22.
7. J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., 1979.
8. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing.* Springer, Berlin, 1997.
9. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.
11. M. Minsky: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
12. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
13. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.
14. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
15. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Lecture Notes in Computer Science*, 6570, in press.
16. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Sci.*, in press.
17. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.
18. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* 3 volumes, Springer, Berlin, 1998.
19. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.
20. The P Systems Website: `http://ppage.psystems.eu`.

Modeling, Verification and Testing of P Systems Using Rodin and ProB

Florentin Ipate, Adrian Ţurcanu

Department of Computer Science, University of Pitesti, Romania

Summary. In this paper we present an approach to modelling, verification and testing for cell-like P-systems based on Event-B and the Rodin platform. We present a general framework for modelling P systems using Event-B, which we then use to implement two P-system models in the Rodin platform. For each of the two models, we use the associated Pro-B model checker to verify properties and we present some of the results obtained.

1 Introduction

Membrane computing, the field initiated by Gheorghe Păun [14], studies computing devices, called P systems, inspired by the functioning and structure of the living cell.

In the last years, the research on various programming approaches related to P systems ([6], [16]) and formal semantics ([4], [2], [7]), or with respect to decidability of some model checking properties [5], has created the need for methods for formally verifying and testing such systems.

Formal verification has been studied for different variants of P systems by using rewriting logic and the Maude tool [2] or, for stochastic systems [3], PRISM and associated probabilistic temporal logic [10]. More recently, NuSMV [9] and Spin [13] have been used to verify various properties of transition P systems. Various approaches to building test cases for such P systems have also been proposed [8], [11], [12].

Event-B is a formal modeling language introduced about 10 years ago by J.R. Abrial [1], used for developing mathematical models of complex systems which behave in a discrete fashion. Event-B is an evolution of the B language, one of the most used modeling language in industry since its introduction in the 90s. The efforts for developing Event-B have been supported by two European research projects: RODIN[1], which produced a first platform for Event-B called *Rodin*, and DEPLOY[2], which is currently enhancing this platform based on feedback from

[1] *http://rodin.cs.ncl.ac.uk* - Project running between 2004-2007
[2] *http://deploy-project.eu* - Project running between 2008-2012

its industrial partners (Bosch, SAP, Siemens and Space Systems Finland), which experiment with the latest development of the platform.

The core technology behind Rodin platform is theorem-proving, but also model-checking (ProB) or animation tools (Anim-B) have been integrated as plug-ins.

In this paper we propose a new approach for verifying and testing transition P systems, based on Event-B and its associated model-checker, ProB. Given the industrial support for Event-B and the strength of the Rodin platform, this approach will have an important impact on the practical use of P systems.

The paper is structured as follows. The next section presents general notions about P systems, the Event-B language, the Rodin platform and the model checker Pro-B. In Section 3 we present a general framework for modeling P systems using Event-B. This is then used to implement two P-system models in the Rodin platform: a simple example in Section 4 and a tritrophic ecosystem in Section 5. For each of the two models, we use the associated Pro-B model checker to verify properties and we present the results obtained. Finally, some conclusions and future work are given in Section 6.

2 Background

2.1 P systems

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. With each region there are associated a finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string $u \in V^*$, where the order is not considered, or by $\Psi_V(u)$. The following definition refers to one of the many variants of P systems, namely cell-like P systems, which use transformation and communication rules [15]. We will call these processing rules. From now onwards we will refer to this model as simply a P system.

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

The membrane structure, μ, is denoted by a string of left and right brackets ([, and]), each with the label of the membrane it points to; μ also describes the position of each membrane in the hierarchy.

The rules in each region have the form $u \rightarrow (a_1, t_1)...(a_m, t_m)$, where u is a multiset of symbols from V, $a_i \in V$, $t_i \in \{in, out, here\}$, $1 \leq i \leq m$. When such a rule is applied to a multiset u in the current region, u is replaced by the symbols a_i with $t_i = here$; symbols a_i with $t_i = out$ are sent to the outer region or outside the system when the current region is the external compartment and symbols a_i with $t_i = in$ are sent into one of the regions contained in the current one, arbitrarily

chosen. In the following definitions and examples all the symbols $(a_i, here)$ are used as a_i. The rules are applied in maximally parallel mode which means that they are used in all the regions at the same time and in each region all the objects to which a rule *can* be applied *must* be the subject of a rule application [14].

Electrical charges from the set $\{+, -, 0\}$ can be also associated with membranes, obtaining P systems with polarizations. In this case, with the same notations from the above definition, we can have many types of rules:

- evolution rules, associated with membranes and depending on the label and the charge of the membranes:
  $[u \to v]_i^p$, $p \in \{+, -, 0\}$, $u \in V, v \in V^*$;
- communication rules, sending an object into a membrane and possibly changing its polarization:
  $u[\,]_i^p \to [v]_i^{fp}$, $p, fp \in \{+, -, 0\}$, $u, v \in V$;
- communication rules, sending an object out of a membrane and possibly changing the polarization of the membrane:
  $[u]_i^p \to [\,]_i^{fp} v$, $p, fp \in \{+, -, 0\}$, $u \in V$, $v \in V \cup \{\lambda\}$

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

2.2 Event-B and Rodin

Event-B is based on set theory as its mathematical foundation. The Event-B models are abstract state machines in which transitions between states are implemented as *events*.

An Event-B model is made of several components. Each component can be either a machine or a context. Contexts contain the static structure of the system: sets, constants and axioms. Axioms define the main properties of sets and constants. On the other hand, machines contain the dynamic structure of the system: variables, invariants, and events. Invariants state the properties of variables and events defines the dynamic of the transition system.

An event is a state transition with the following simplified structure:

Event *eventName*
refines <list of refined events (if any) >

when
  grd1 :
    ⋮
  grdn :
then

```
    act1 :
        ⋮
    actn :
end
```

Guards (*grd1*, ..., *grdn*) are necessary conditions for an event to be enabled. They are theorems derivable from invariants, axioms and previously declared guards. An event may have no guards; in this case it is permanently enabled.

Actions (*act1*, ..., *actn*) describe how the occurrence of an event will modify some of the variables of the machine. All actions of an event are performed at the same time. An action might be either deterministic (using the normal assignment operator :=) or non-deterministic. Non-deterministic actions use the :∈ operator; they have the form $x :\in \{$ *set of possible values* $\}$, in which case an arbitrarily chosen value from the set of possible values is assigned to the variable x.

A very important Event-B concept is refinement, which allows a model to be developed gradually. When the model is finished, a Rodin Platform tool, called Proof Obligation Generator, decides what is to be proved in order to ensure the correctness of the model (e.g. invariant preservation, consistency between original and refined models). Therefore, the proving mechanism provides the guarantee of a formally correct model before the model checker is actually used. This is a big strength of the Rodin platform

2.3 ProB - more than just another model checker

ProB is an animation and model checking tool which accepts B-models, but is also integrated within the Rodin platform. Unlike, most model checking tools, ProB works on higher-level formalisms and so it enables a more convenient modeling.

Properties of an Event-B model can be verified using either the ProB version within the Rodin platform or the standalone version, which offers a greater range of facilities, such as computation of operation coverage or the possibility to find states satisfying a predicate or enabling an operation. When the standalone version is used, the model can be automatically translated in the B language and imported into ProB.

ProB supports automated consistency checking, which can be used to detect various errors in B specifications. The animation facilities allow: to visualize, at any moment, the state space, to execute a given number of operations, to see the shortest trace to current state. Properties that are intended to be verified can be formulated using the LTL or the CTL formalism.

3 The Event-B model of a P system

In this section we present the main ideas about how to build the Event-B model of a P system.

For each object x that appears on the left side of a rule, we introduce a variable xc, representing the number of objects of that type that can be consumed. When the rule is applied, this variable is decreased accordingly. For each object x that appears on the right side of a rule, we introduce a variable xp representing the number of produced objects of that type. When the rule is applied, the variable is incremented. Furthermore, in multi-membrane systems, variables are indexed by membrane numbers.

For each rule we introduce an event. The event is enabled if the rule can be applied and its application modify the state of the system accordingly. A special event, called **actualization**, that is enabled after each step of maximal parallelism, is also needed in order to update the variables before the next computation step.

The initial model can then be refined by adding details about the *state* of the computation. At the beginning of every step of maximal parallelism, the system is considered to be in state *Running*. Another state, *Other*, is considered as an intermediate state between two such steps. A halting configuration is marked by a transition from state *Running* to another state, $Halt$. Finally, in order to keep the number of configurations under control, we can assume that each component of a configuration cannot exceed an established upper bound (denoted MAX) and also that each rule can only be applied for at most a given number of times (denoted SUP). When either of these conditions is violated, we consider that the system performs a transition from *Running* to a fourth state $Crash$. All these states are implemented using a variable, called *state*, with four possible values: *Running*, *Other*, $Halt$ and $Crash$. Obviously, all the events in the original model have to be refined - these now become transitions between states *Running*, and *Other*. Furthermore, new events have to be introduced for transitions from *Running* to $Halt$, *Running* to $Crash$, $Halt$ to $Halt$ and $Crash$ to $Crash$. Obviously, the *state* variable and the extra transitions could have been introduced directly in the original model. However, the use of refinement allows a gradual, more manageable and natural, construction of the model.

4 A simple example

We consider as first example a P system with one membrane and four rules: $\Pi_1 = \{V = \{s, a, b, c\}, []_1, w_1 = s, R = \{r_1 : s \to ab,\ r_2 : a \to c,\ r_3 : b \to bc,\ r_4 : b \to c\}\}$.

We build the corresponding Event-B model in two steps: first, we are interested only of its evolution, then we refine it by introducing the variable **state** presented before.

The first model is just a machine with six variables, all natural numbers (sc, ac, ap, bc, bp, cp) and six events: the initialization event, four events (each of them corresponding to a rule) and the actualization event.

For example, the event corresponding to the first rule and the actualization event are as follows:

Event *rule1*

when
grd1 : $sc > 0$
then
act1 : $sc := sc - 1$
act2 : $ap := ap + 1$
act3 : $bp := bp + 1$
end

and respectively,

Event *actualization*

when
grd1 : $sc + ac + bc = 0$
grd2 : $ap + bp + cp > 0$
then
act1 : $ac := ap$
act2 : $bc := bp$
act3 : $ap := 0$
act4 : $bp := 0$
end

For this model, 20 proof obligations are generated and automatically checked. Using the associated model checker ProB, we verify our formally correct model and we discover that, the sequence of events *rule1, rule2, rule4* leads to the permanent enabling of the actualization event. In order to fix this problem, we refine our model by introducing the variable **state**. Obviously, all events in the original model need to be refined. For example, the refinement of the event *rule1* is as follows:

Event *rule1*
refines *rule1*

when
grd1 : $n1 < SUP$
grd2 : $sc > 0$
grd3 : $ac + ap < MAX$
grd4 : $bc + bp < MAX$
then
act1 : $n1 := n1 + 1$
act2 : $sc := sc - 1$
act3 : $ap := ap + 1$
act4 : $bp := bp + 1$
act5 : $state := Other$
end

We also introduce new events, corresponding to the transitions between the states of the computation, such as:

Event *RunToCrash*

when
grd1 : $state = Running$
grd2 : $(s > 0 \wedge n1 = SUP) \vee (ac > 0 \wedge n2 = SUP) \vee (bc > 0 \wedge (n3 = SUP \vee n4 = SUP)) \vee (ac = MAX) \vee (bc = MAX) \vee (cp = MAX)$
then
act1 : $state := Crash$
end

and

Event *RunToHalt*

when
grd1 : $state = Running$
grd2 : $sc + ac + bc = 0$
then
act1 : $state := Halt$
end

In this case, there are 60 proof obligations generated, all automatically proven. Using the model checking facilities available in the Rodin platform, we verify the consistency of our model and we find no deadlocks or invariant violation.

Once a formally proven model of the P system is in place, we can use the formalism to verify its properties or to generate tests for certain coverage criteria [12]. The underlying idea of testing using model checkers is to formulate the coverage criterion as a temporal logic formula, negate it and interpret counterexamples (returned by ProB) as test cases. For example, a counterexample for the (negated) LTL formula $G\{not(n4 > 0)\ or\ state = Other\}$ is a test case which covers rule4.

Some examples of properties, their truth values and counterexample returned (for false properties) are given in Table 1.

Note that, in this table, the symbol "/ =" means "≠" and the values for the two constants MAX and SUP were both considered to be 10.

5 Modeling an Ecosystem

We consider now a more complex example: a P system Π_2 with two membranes and electrical charges, which models a tritrophic ecosystem. Its alphabet is $V = \{C, H, P, b, cycle1, cycle2, cycle3, cycle4, g, s\}$, where C stands for carnivores, H for herbivores, P for plants, b for bones, g for garbage and s for volatile substances that attracts carnivores. The initial multiset is P^{100} in the first membrane and H^{300}, C^{10} and $cycle1$ in the second one.

The ecosystem evolves in four cycles:

- Cycle1: Reproduction of plants
  - rule14: $P^2[\ ]_2 \rightarrow [P^3]$ some plants reproduce
  - rule15: $P^2[\ ]_2 \rightarrow [P^2]$ other plants do not reproduce

LTL Property	Truth Value
$G\{ac \in \{0,1\}\ or\ state = Other\}$	True
$F\{cp > 2\ \&\ state/ = Other\}$	False rule1 rule2 rule4 RunToHalt
$\{cp = 0\ \ or\ \ state = Other\}$ U $\{cp = 2\ \&\ state/ = Other\}$	True
$G\{(state = Running \Rightarrow cp <= MAX)$ or $(state = Other)\}$	True
$G\{not(n4 > 0)\ or\ state = Other\}$	False rule1 rule2 $(\text{rule3})^8$ rule4
$F\{state = Halt\}$	False rule1 rule2 $(\text{rule3})^9$ RunToCrash
$G\{((n2 > 0\ \&\ n3 > 0) => cp >= bc)$ $or\ (state = Other)\}$	True

Table 1. Properties checked for Π_1

  - rule7: $[cycle1 \rightarrow cycle2]_2$ transition to cycle2
- Cycle2: Herbivores alimentation
  - rule8: $[HP]_2 \rightarrow +[H^2s]g$ some herbivores feed themselves and produce other herbivores, a volatile substance that attract carnivore and some garbage
  - rule9: $[HP]_2 \rightarrow +[HP]g$ other herbivores do not feed
  - rule10: $[cycle2]_2 \rightarrow +[cycle3]g$ transition to cycle3
- Cycle3: Carnivores alimentation
  - rule11: $+[CHs]_2 \rightarrow -[C^2]g$ some carnivores feed themselves and produce other carnivores and some garbage
  - rule12: $+[CHs]_2 \rightarrow -[CHs]g$ other carnivores do not feed
  - rule13: $+[cycle3]_2 \rightarrow [cycle4]g$ transition to cycle4
- Cycle4: Mortality and reinitialization
  - rule1: $-[P]_2 \rightarrow [\ ]P$ all the plants survive to the next cycle
  - rule2: $-[H]_2 \rightarrow [b]g$ some herbivores die
  - rule3: $-[H]_2 \rightarrow [H]g$ others survive to the next cycle
  - rule4: $-[C]_2 \rightarrow [b]g$ some carnivores die
  - rule5: $-[C]_2 \rightarrow [C]g$ others survive to the next cycle
  - rule6: $-[cycle4]_2 \rightarrow [cycle1]g$ reinitialization
  - rule16: $[g \rightarrow \lambda]_1$ elimination of garbage

For clarity of presentation, in this case we build the model in one step, introducing from the beginning the variable **state**, but without the "crash" situation. When a P system uses electrical charges, at each step of maximal parallelism, only rules that have the same initial and final polarization can be applied. In order to implement this requirement, we use two variables **polarization2** and **fp2** for the initial, and, respectively, for the final polarization of the second membrane. Besides the usual polarization values (*plus*, *minus* and *zero*), we introduce an in-

termediate value, *all*, that we use as an initial value for **fp2**. In the **actualization** event, **fp2** is reset to the value *all* and *polarization*2 receives the value of **fp2**.

As an example, the event corresponding to rule 1 is presented below.

Event *rule1*

when
grd1 : $p2c \geq 1$
grd2 : $polarization2 = minus$
grd3 : $fp2 = all \vee fp2 = zero$
then
act1 : $p2c := p2c - 1$
act2 : $p1p := p1p + 1$
act3 : $n1 := n1 + 1$
act4 : $fp2 := zero$
act5 : $state := Other$
end

The model checker ProB can then be used to verify ecosystem properties. Table 2 summarizes some of these properties, along with the result produced by the model checker. For all the properties, we considered 5000 as the maximum number of new states.

LTL property	Truth Value
$G\{p2c > 10 \; or \; state = Other\}$	False
$F\{c2c = 0 \; \& \; state/ = Other\}$	True
$F\{p2c = 0 \; \& \; state/ = Other\}$	True
$F\{p2c = 100 \; \& \; state/ = Other\}$	False
$F\{(cycle12c = 1 \; \& \; state/ = Other) => g1c = 0\}$	True

Table 2. Properties checked for the ecosystem Π_2

If we manually execute the initialization event and we choose to see the state space then the result obtained is as shown in Figure 1. Therefore, we have three enabled events - rule14, rule15 and rule7 - and we can see what it happens if we choose each of them.

Another option allows us to verify if the model contains deadlocks or invariant violations. After 20136 new states explored in 93 seconds the coverage analysis shows that the events **RunToHalt** and **HaltToHalt** are not covered.

6 Conclusions and future work

Event-B, Rodin and ProB are not just another modeling language, platform and model checker. Based on rigorous mathematical foundation and allowing high-level modeling, they are strongly supported by the industry.

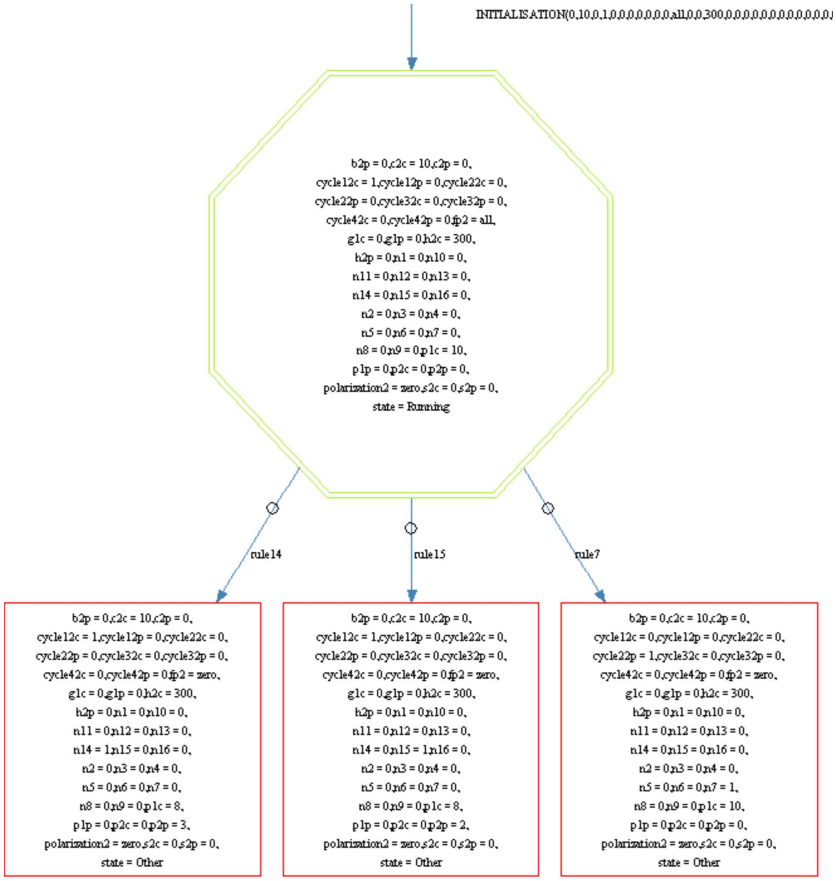


Fig. 1. The state space shown by ProB

In this paper we have presented a general framework for modeling P systems using Event-B and applied the proposed approach on two examples.

Our future work will concentrate on modeling other types of P systems, refinement, simplification, decomposition of models, as well as applying search based techniques for test generation.

Acknowledgement

Adrian Ţurcanu was supported by project POSDRU -"Dezvoltarea scolilor doctorale prin acordarea de burse tinerilor doctoranzi cu frecventa"- 88/1.5/S/52826 and Florentin Ipate was partially supported by project Deploy, EC-grant no. 214158, and Romanian Research Grant CNCSIS-UEFISCDI no. 7/05.08.2010 at the University of Pitesti.

References

1. Abrial, J.R., Modeling in Event-B. System and software engineering, Cambridge University Press, (2010).
2. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theor. Comput. Sci. 373(3), 163–181 (2007)
3. Bernardini, F., Gheorghe, M., Romero-Campero, F.J., Walkinshaw, N.: A hybrid approach to modeling biological systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 8th International Workshop, WMC 2007, Revised Selected and Invited Papers. LNCS, vol. 4860, pp. 138–159. Springer (2007)
4. Ciobanu, G.: Semantics of P systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, chap. 16, pp. 413–436. Oxford University Press (2010)
5. 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)
6. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for P systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, chap. 17, pp. 437–454. Oxford University Press (2010)
7. Kleijn, J., Koutny, M.: Petri nets and membrane computing. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, chap. 15, pp. 389–412. Oxford University Press (2010)
8. Gheorghe, M., Ipate, F.: On testing P systems. 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. LNCS, vol. 5391, pp. 204–216. Springer (2009)
9. Gheorghe, M., Ipate, F., Lefticaru, R., Dragomir, C. , An integrated approach to P systems formal verification, in *Proc. 11th Int. Conf. on Membrane Computing*, eds. M. Gheorghe, T. Hinze and Gh. Păun, 225–238, ProBusiness Verlag, Berlin (2010)
10. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer (2006)
11. Ipate, F., Gheorghe, M.: Testing non-deterministic stream X-machine models and P systems. Electronic Notes in Theoretical Computer Science 227, 113–126 (2009)
12. Ipate, F., Gheorghe, M., Lefticaru, R., Test generation from P systems using model checking, *J. Logic Algebr. Program.* 79(6), 350–362 (2010)
13. Ipate, F., Lefticaru, R., Tudose, C., Formal Verification of P Systems Using SPIN, Int. J. Found. Comput. Sci. 22(1): 133-142 (2011)
14. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
15. Păun, G.: Membrane Computing: An Introduction. Springer-Verlag (2002)
16. Serbanuta, T., Stefanescu, G., Rosu, G.: Defining and executing P systems with structured data in K. 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. LNCS, vol. 5391, pp. 374–393. Springer (2009)

Forward and Backward Chaining with P Systems

Sergiu Ivanov[1,2], Artiom Alhazov[1,3], Vladimir Rogojin[1,4],
Miguel A. Gutiérrez-Naranjo[5]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: {sivanov,artiom}@math.md
[2] Technical University of Moldova, Faculty of Computers,
Informatics and Microelectronics,
Ştefan cel Mare 168, Chişinău MD-2004 Moldova
[3] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
[4] Research Programs Unit, Genome-Scale Biology,
Faculty of Medicine, Helsinki University,
Biomedicum, Haartmaninkatu 8, Helsinki 00014, Finland
E-mail: vladimir.rogojin@helsinki.fi
[5] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: magutier@us.es

Summary. On the one hand, one of the concepts which lies at the basis of membrane computing is the multiset rewriting rule. On the other hand, the paradigm of rules is profusely used in computer science for representing and dealing with knowledge. Therefore, it makes much scene to establish a "bridge" between these domains, for instance, by designing P systems reproducing forward and backward chaining which can be used as tools for reasoning in propositional logic. Our work shows again, how powerful and intuitive the formalism of membrane computing is and how it can be used to represent concepts and notions from totally unrelated areas.

1 Introduction

The use of rules is one of the most common paradigms in computer science for dealing with knowledge. Given two pieces of knowledge V and W, expressed in some language, the rule $V \to W$ is usually considered as a causal relation between V and W. This representation is universal in science. For example, in chemistry, V and W can be metabolites and $V \to W$ a chemical reaction. In this case,

V represents the reactants which are consumed in the reaction and W is the obtained product. In ecology, W may represent the population obtained from the set of individuals V after a time unit. In computer science, V and W are pieces of information (usually split into unit pieces $v_1, v_2, \ldots, v_n$ and $w_1, w_2, \ldots, w_m$) and the rule $V \to W$ is the representation of the precedence relation between V and W. In propositional logic, $V \to W$ is a representation of the clause $\neg v_1 \neg v_2 \vee \cdots \vee \neg v_n \vee w_1 \vee w_2 \vee \cdots \vee w_m$.

Besides representing knowledge using this paradigm, scientists are interested in the derivation of new knowledge from a known piece of information: given a knowledge base $KB = (A, R)$, where A is a set of known atoms and a set R of rules of type $V \to W$, the problem is to know if a new atom g can be obtained from the known atoms and rules. We will call this problem a *reasoning problem* and it will be denoted by $\langle A, R, g \rangle$.

In computer science, there are two basic method for seeking a solution of a reasoning problem, both of them based on the inference rule know as Generalized Modus Ponens: the former is data-driven and it is known as *forward chaining*, the latter is query-driven and it is called *backward chaining*.

In this paper we will consider knowledge bases on propositional logic and prove that both types of chaining can be simulated by P systems. In this way, given a reasoning problem we present several methods for building P systems Π_b and Π_f which produce the objects YES or NO if and only if the corresponding chaining method gives a positive or negative answer.

As one should observe, even though logic inference rules and multiset rewriting rules originate from totaly different areas of mathematics and computer science and represent unrelated notions, their concepts have some similarities. In particular, no information about the ordering of elements in both left and right sides of the rules of both types is used. On the other hand, the inference rules could be thought of as set rewriting rules, while multiset rewriting rules operate at multisets. However, multiset rewriting rules could be interpreted as set rewriting rules if one ignores the multiplicity of elements of the multiset. Therefore we could represent sets of facts in P systems as multisets of objects and inference rules as multiset rewriting rules. When one considers the set of facts represented in a region of a P systems, one only considers the underlying set of the region's multiset.

The paper is organized as follows. First we recall some basic definitions related to the reasoning problem. Then we present our constructions and prove that the obtained P systems produce the same answer as the corresponding chainings. Finally, some conclusions and open research directions are proposed.

2 Definitions

2.1 Transitional P Systems

A *transitional* membrane system is defined by a tuple

$\Pi = (O, \mu, w_1, w_2, \cdots, w_m, R_1, R_2, \ldots, R_m, i_0)$, where

O is a finite set of objects,

μ is a hierarchical structure of m membranes, bijectively labeled by $1, \ldots, m$; 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; in this paper i_0 is the skin and could be omitted.

The rules of a membrane systems have the form $u \to v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$ are written as a subscript, and target *here* is typically omitted. In case of non-cooperative rules, $u \in O$. In this paper we will not consider target indications.

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

A sequence of transitions is called a computation. The computation halts when such a configuration is reached that no rules are applicable. Since in this paper we will focus on deciding P systems, we are only interested in the presence of one of the special symbols $\{\text{YES}, \text{NO}\}$ in the halting configuration of a computation.

In transitional P systems with promoters/inhibitors we consider rules of the following forms:

- $u \to v|_a$, $a \in O$ – this rule is only allowed to be applied when the membrane it is associated with contains at least an instance of a; a is called the *promoter* of this rule;
- $u \to v|_{\neg a}$, $a \in O$ – this rule is only allowed to be applied when the membrane it is associated with contains no instances of a; a is called the *inhibitor* of this rule.

Rules do not consume the corresponding promoters/inhibitors. A rule may have both a promoter and an inhibitor at the same time, in which case it can only be applied when there is at least one instance of the promoter *and* no instances of the inhibitor in the region. Note also, that a single instance of an object may act as a promoter for more than one instance of rewriting rules during the same transition.

2.2 P Systems with Active Membranes

A P system with *active membranes* is defined by a tuple

$\Pi = (O, H, \mu, w_1, w_2, \ldots, w_m, R, i_0)$, where

O is a finite set of objects,

H is the alphabet of names of membranes,

μ is the initial hierarchical structure of m membranes, bijectively labeled by $1, \ldots, m$;

w_i is the initial multiset in region $i, 1 \leq i \leq m$,

R is the set of rules,

i_0 is the output region; in this paper i_0 is the skin and could be omitted.

The rules in P systems with active membranes can be of the following five basic types:

(a) $[a \to v]_h^e$, $h \in H$, $e \in \{+, -, 0\}$, $a \in O$, $v \in O^*$; in this paper we consider the extension $a \in O^*$;
(b) $a[\,]_h^{e_1} \to [b]_h^{e_2}$, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $a, b \in O$;
(c) $[a]_h^{e_1} \to [\,]_h^{e_2} b$, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $a, b \in O$;
(d) $[a]_h^e \to b$, $h \in H \backslash \{s\}$, $e \in \{+, -, 0\}$, $a, b \in O$;
(e) $[a]_h^{e_1} \to [b]_h^{e_2} [c]_h^{e_3}$, $h \in H \backslash \{s\}$, $e_1, e_2, e_3 \in \{+, -, 0\}$, $a, b, c \in O$.

The rules apply to elementary membranes, i.e. membranes which do not contain other membranes inside.

The rules are applied in the usual non-deterministic maximally parallel manner, with the following details: any object can be subject of only one rule of any type and any membrane can be subject of only one rule of types (b)–(e). Rules of type (a) are not counted as applied to membranes, but only to objects. This means that when a rule of type (a) is applied, the membrane can also evolve by means of a rule of another type. If a rule of type (e) is applied to a membrane, and its inner objects evolve at the same step, it is assumed that first the inner objects evolve and then the division takes place, so that the result of applying rules inside the original membrane is replicated in the two new membranes.

2.3 Formal Logic Preliminaries

Definition 1. *An* atomic formula *(also called an* atom*) is a formula with no deeper structure.*

An atomic formula is used to express some fact in the context of a given problem. The *universal set* of atoms is denoted with U. For a set A, $|A|$ is the number of elements in this set (cardinality).

Definition 2. *A* knowledge base *is a construct* $KB = (A, R)$ *where* $A = \{a_1, a_2, \ldots, a_n\} \subseteq U$ *is the set of known atoms and* R *is the set of rules of the form* $V \to W$*, with* $V, W \subseteq U$.

In propositional logic, the *derivation* of a proposition is done via the inference rule known as Generalized Modus Ponens:

$$\frac{P_1, P_2, \ldots, P_n \ , \quad P_1 \wedge P_2 \wedge \cdots \wedge P_n \rightarrow Q}{Q}$$

The meaning of this rule is as follows: if $P_1 \wedge P_2 \wedge \cdots \wedge P_n \rightarrow Q$ is a known rule and $P_1, P_2, \ldots, P_n \subseteq A$ then, Q can be derived from this knowledge. Given a knowledge base $KB = (A, R)$ and an atomic formula $g \in U$, we say that g can be derived from KB, denoted by $KB \vdash g$, if there exists a finite sequence of atomic formulas $F_1, \ldots F_k$ such that $F_k = g$ and for each $i \in \{1, \ldots, k\}$ one of the following claims holds:

- $F_i \in A$.
- F_i can be derived via Generalized Modus Ponens from R and the set of atoms $\{F_1, F_2, \ldots, F_{i-1}\}$

It is important to remark that for rules $V \rightarrow W$ we can require $|W| = 1$ without losing generality. Indeed, $V \rightarrow W = \neg V \vee W$. If $W = w_1 \wedge w_2 \wedge ... \wedge w_n$,

$$V \rightarrow W = \neg V \vee \bigwedge_{i=1}^{n} w_i = \bigwedge_{i=1}^{n} \neg V \vee w_i = \bigwedge_{i=1}^{n} V \rightarrow w_i$$

This conclusion also makes it clear how to transform set of rules R to R' with the property that all right-hand sides contain no more than one symbol.

This definition of derivation provides two algorithms to answer the question of knowing if an atom g can be derived from a knowledge base KB. The first one is known as *forward chaining* and it is an example of data-driven reasoning, i.e., the starting point is the known data. The dual situation is the *backward chaining*, where the reasoning is query-driven.

Definition 3. Forward chaining *decides $KB \vdash g$ by constructing the closure K of the set of known facts A under the operation of adding new facts to the set by applying Generalized Modus Ponens and checking $g \in K$.*

Definition 4. Backward chaining *decides $KB \vdash g$ by attempting to find some resolution of the goal fact g to the set of known facts A by substituting the right-hand sides of the rules with their corresponding left-hand sides.*

We will call *reduction* the substitution of the right-hand side of a rule with the corresponding left-hand side in the process of backward chaining.

A deep study of both algorithms is out of the scope of this paper. We briefly recall their basic forms.

Forward chaining
INPUT: A reasoning problem $\langle A, R, g \rangle$
INITIALIZE: $Deduced = A, Deduced' = \emptyset$
while $Deduced \neq Deduced'$ **do**

```
    Deduced' ← Deduced
    for all (P1P2...Pn → Q) ∈ R such that {P1, P2, ..., Pn} ⊆ Deduced' do
      if Q = g then
        return true
      else
        Deduced ← Deduced ∪ {Q}
      end if
    end for
  end while
  if g ∉ Deduced then
    return false
  end if
```

Backward chaining

INPUT: A reasoning problem $\langle A, R, Targets\rangle$

```
  if Targets = ∅ then
    return true
  else
    Found ← false
    Actual ← SelectOne(Targets)
    for all (P1P2...Pn → Actual) ∈ R do
      NewTargets ← ({P1, P2, ..., Pn}\A) ∪ (Targets\{Actual})
      Found ← Found ∨ (Backward Chaining⟨A ∪ {Actual}, R, NewTargets⟩)
    end for
    return Found
  end if
```

Instead of only having one goal in the input of the backward chaining algorithm, we consider a set of goal facts $Targets$. In the case of only one goal fact, this set is initially $\{g\}$. Note that this algorithm does not always produce a correct result in the cases when the inference rules form cycles like, for example, $\{a \to b, b \to a\}$.

In this paper we present several different transformations of a tuple $\langle A, R, g\rangle$ into P systems and prove than forward chaining and backward chaining can be represented and performed in the usual semantics of membrane computing. We will write multisets in string notation. We will use the symbol $(\cdot)$ to denote multiset union.

The problem of deriving a new piece of knowledge from given ones and how such derivation can be made automatically has been studied for centuries. In this paper we explore a small part of this problem. In other logic systems, as relational logic, clausal logic, first or higher order logic, many other problems as the unification of terms must be considered. We refer the interested reader to [1].

In the paper, $v_1v_2\ldots v_n$ may mean either a conjunction of atoms in an inference rule or a multiset of objects representing such a conjunction. Which of these is actually meant should be clear from the context.

3 Forward Chaining

Let us consider the reasoning problem $\langle A, R, g\rangle$. Forward chaining basically consists in finding all facts that can be derived from A according to R and checking whether g is among these facts.

We will now try to design a transitional P system which will implement forward chaining. We will focus on constructing a non-uniform solution, because in this way we will be able to map inference rules directly to multiset rewriting rules in P systems.

Intuitively, the forward chaining algorithm consists of successive application of rules. All rules can be applied in any order and produce the same result. This leads us to the conclusion that the standard maximally parallel strategy of applying rules in P systems is suitable for carrying out forward chaining.

In this first approach we will look at the propositional rule $V \to W$ from our knowledge base as an evolution rule of a P system where all the objects in both sides of the rule have multiplicity one. Before we start, we need to introduce some considerations.

- First of all, in P systems the objects in the LHS of the rule are consumed when the rule is applied. This is a serious drawback for a direct translation of propositional logic into P systems. This limitation can be avoided if we introduce a copy of the LHS into the RHS of the rule, thus considering a multiset rewriting rule $V \to VW$ for each propositional rule $V \to W$.
- Copying the LHS into the RHS introduces new undesirable effects. One of them is that a rule can be applied indefinitely many times, since the objects which trigger the rule will be in the membrane forever. This can be also avoided by introducing a new object γ_i for each propositional rule $r_i \equiv V \to W$ and adding it to the LHS of the membrane computing rule $\gamma_i V \to VW$. This object γ_i is consumed and allows the rule to be applied at most once.
- The answer YES can be easily produced by using a rule $g \to YES$. As soon as g is generated, the object YES is produced. The answer NO should be obtained if new atoms can be deduced. From a membrane computing point of view, it is not so easy to check if a membrane has a new object different from the previous configuration, but we can consider an upper bound on the number of steps in order to check if g has been produced or not. This upper bound is related to the number of rules, since each rule can be only applied once.

We construct a P system implementing chaining according to the remarks given above:

$$
\begin{aligned}
\Pi_0 &= (U_0, [\ ]_s, w_s^{(0)}, R_0, s), \text{ where}\\
U_0 &= U \cup \{\text{YES}\} \cup \{\gamma_i \mid 1 \le i \le n\},\\
w_s^{(0)} &= \gamma_1\gamma_2 \dots \gamma_n,\\
R_0 &= \{\gamma_i V \to VW \mid (r_i : V \to W) \in R, 1 \le i \le, n\} \cup \{g \to \text{YES}\},\\
n &= |R|.
\end{aligned}
$$

Note that labeling the inference rules in R is done injectively.

We placed the initial set of facts A into the skin membrane and let some multiset rewriting rules easily obtained from R to simulate the forward-chaining inference process according the set of inference rules R. The rule $g \to \text{YES}$ is waiting for the goal to appear in the region. As soon as the goal appears, the rule produces a YES-object.

Π_0 is very simple and illustrates vividly how easily very basic forward chaining can be done in P systems. Π_0 always stops, and there is a YES in w_s when g can be derived from the facts in A.

To place a NO into the skin at proper times requires a further observation that the upper bound on the number of steps Π_0 makes is $n+1$. Indeed, all rules in R_0 may be applied only once and $|R_0| = |R| + 1 = n + 1$. Thus, we may wait until all the rules in the system are exhausted. If after $n+1$ steps the symbol YES has not been produced, the system should produce a NO. In the following P system Π_1 we have implemented the timer:

$$
\begin{aligned}
\Pi_1 &= (U_1, [\ ]_s, w_s^{(1)}, R_1, s), \text{ where} \\
U_1 &= U_0 \cup \{t_i \mid 0 \le i \le n+1\} \cup \{\text{NO}\}, \\
w_s^{(1)} &= w_s^{(0)} \cdot t_0, \\
R_1 &= R_0 \cup \{t_i \to t_{i+1} \mid 0 \le i \le, n\} \cup \{t_{n+1} \to \text{NO}, \text{YES NO} \to \text{YES}\}.
\end{aligned}
$$

Π_1 will always stop in either $n+1$ steps if a NO has been produced, or in $n+2$ steps it a YES has been produced. To nondeterministically minimize the number of steps, one may consider $R_1' = R_1 \cup \{t_i \text{YES} \to \text{YES} \mid 1 \le i \le n\}$. This, however, does not guarantee that the system will stop in a small (constant) number of steps after a YES has been produced and, in the worst case, it possible that the whole chain of transformations of t_i will take place.

To assure that the system always stops when no rules are being applied, we can use rules with inhibitors. Consider the following P system:

$$
\begin{aligned}
\Pi_2 &= (U_2, [\ ]_s, w_s^{(2)}, R_2, s), \text{ where} \\
U_2 &= U_0 \cup \{t, p, \text{NO}\}, \\
w_s^{(2)} &= w_s^{(0)} \cdot tp, \\
R_2 &= \{\gamma_i V \to VWp|_{\neg \text{YES}} \mid (r_i : V \to W) \in R, 1 \le i \le n\} \cup \\
&\quad \cup \{p \to \lambda, t \to \text{NO}|_{\neg p}, gt \to \text{YES}\}.
\end{aligned}
$$

Any rule application produces an instance of p, which is immediately erased. While rules are still being applied, p is always present in the system and thus t cannot change into NO. When rules are not being applied any more, p is erased from the system and t evolves into NO. If a rule application adds the goal symbol g to the system, g consumes t and produces YES. Thus, when no more rules can be applied, the system always needs two more steps to produce a NO. When the goal fact is produced, the system always needs one more step to produce a YES.

The inhibitors are required when, for the problem $\langle A, R, g\rangle$, $\exists(V \to W) \in R$ such that $g \in V$. Once YES is present in the system, computations should stop.

The last problem to solve is cleaning up. This is pretty obvious:

$$\Pi_f^{(1)} = \Pi_3 = (U_2, [\ ]_s, w_s^{(2)}, R_3, s), \text{ where}$$
$$R_3 = R_2 \cup \{a \to \lambda|_{\neg p} \mid a \in U \cup \{\gamma_i \mid 1 \le i \le n\}\}.$$

When the system produces a YES, the application of rules derived from R stops and p is not produced any more. This allows the rules $a \to \lambda|_{\neg p}$ to clean everything in one extra step. When there are no more rules derived from R to apply, p is not produced as well, which triggers the clean-up. Note that the clean-up procedure does not considerably alter the number of steps Π_3 needs to solve the problem.

Π_3 takes advantage of the maximal parallelism and always applies as many rules as possible at the same time. However, if there are several ways to derive g from A, Π_3 may not always follow the most efficient strategy.

4 A Different Approach to Forward Chaining

We will now try to go beyond the most trivial translation of a decision problem to a P system. We will consider a P system $\Pi_f^{(2)}$ with a single membrane, and a set of rules of type $v \to w|_{\neg i;p}$ where i, p, v, w are objects of the alphabet.

We will translate a rule $r_i \equiv u_1 u_2 \dots u_n \to v$ from R into n rules: $\rho_{ij} \equiv r_{ij} \to r_{ij+1}|_{\neg v;u_j}$ for $j \in \{1, \dots, n-1\}$ and $\rho_{in} \equiv r_{in} \to v|_{\neg v;u_j}$. Note that we require that the right-hand side of every rule in R should contain exactly one fact. We will also add the following rules to implement the timer:

$$\{t_k \to t_{k+1}|_{\neg g;t_k} \mid 1 \le k \le l-1\} \cup \{t_k \to \text{YES}|_{\neg \text{NO};g} \mid 0 \le k \le l\}$$
$$\cup \{t_l \to \text{NO}|_{\neg g;t_l}\}$$

The following rules will clean up the regions of the system:

$$\{a \to \lambda|_{\neg \text{NO};\text{YES}} \mid a \in \Gamma \backslash \{\text{YES}\}\} \cup \{b \to \lambda|_{\neg \text{YES};\text{NO}} \mid b \in \Gamma \backslash \{\text{NO}\}\}$$

Here g is the goal fact and Γ is the alphabet of the P system. l is the sum of the lengths of the left-hand sides of all rules in R. In other words, l is the maximal number of steps $\Pi_f^{(2)}$ has to go through to try all rules from R.

The alphabet contains all the atoms from U, the symbols $\{\text{YES}, \text{NO}\}$, all t_k, $1 \le k \le l$, and all the r_{ij} where i is the index of the corresponding rule from R and j is the index of an atom in the LHS of the rule r_i.

In the initial configuration the skin membrane contains all objects from A, an object t_0, and all objects r_{i1}, $1 \le i \le |R|$.

The rules ρ_{ij} are meant to check whether all left-hand-side symbols of the rule r_i are present in the system. If this condition is satisfied, the right-hand side of the rule r_i is added. In parallel with the application of rules ρ_{ij} the timer symbols

t_k evolve from t_1 to t_l. If the goal symbol g is produced, the rule $t_k \to \text{YES}|_{\neg\text{NO};g}$ produces YES. This will lead to the eventual erasure of all other symbols. If, however, the goal symbol is not produced before t_l appears in the system, a NO is produced and forces the erasure of all other symbols.

Example 1. Consider the tuple $\langle A, R, d\rangle$ with $A = \{a, b\}$ and $R = \{r_1 \equiv ab \to c, r_2 \equiv bc \to d\}$. From this deduction problem we can construct the P system $\Pi = (\Gamma, w, R_f)$ where

- the alphabet is $\Gamma = \{a, b, c, d, r_{11}, r_{12}, r_{21}, r_{22}, t_1, t_2, t_3, t_4, \text{YES}, \text{NO}\}$;
- the initial multiset in the unique membrane is $w = abr_{11}r_{21}t_0$;
- the rules ρ_{ij} are:

$$\rho_{11} \equiv r_{11} \to r_{12}|_{\neg c;a} \qquad \rho_{21} \equiv r_{21} \to r_{22}|_{\neg d;b}$$
$$\rho_{12} \equiv r_{12} \to c|_{\neg c;b} \qquad \rho_{22} \equiv r_{22} \to d|_{\neg d;c}$$

In the initial configuration $C_0 = [abr_{11}r_{21}t_0]$ rules ρ_{11} and ρ_{21} can be applied, which yields the configuration $C_1 = [abr_{12}r_{22}t_1]$. In C_1 the rule ρ_{12} can be applied and we obtain $C_2 = [abcr_{22}t_2]$. Now, by applying ρ_{22} we obtain $C_3 = [abcdt_3]$. Since the goal fact has appeared in the system, t_3 will evolve into a YES: $C_4 = [abcd\text{YES}]$. In the next step all symbols but YES will be erased: $C_5 = [\text{YES}]$.

One of the main differences between the usual semantics in P systems and the semantics in propositional logic is that the application of a rule in membrane computing consumes the objects in the LHS of the rule. This is undesirable from the point of view of propositional logic, since the validity of an atom does not change if the atom is used in a derivation. This drawback is avoided by using new auxiliary objects r_{ij} which are consumed instead of atoms.

Using these auxiliary objects has other positive effects as well. In propositional logic, once the rule $a \to b$ has been used to derive b, the rule will not be used any more. Or rather, further applications of this rule make no difference, since in propositional logic rules operate on sets of facts. This property needs to be treated specially in P systems since we use maximal parallelism and if a rule can be applied multiple times it will be applied so. By consuming the objects r_{ij} we avoid multiple applications of rules.

Finally, the use of inhibitors stops the production of an object (the derivation of an atom) if this object has been previously produced by another rule.

Theorem 1. *$\Pi_f^{(2)}$ solves the reasoning problem it was designed for using forward chaining.*

Proof. $\Pi_f^{(2)}$ works by transforming the symbols r_{i1} into the corresponding right-hand sides of rules $(r_i \equiv V_i \to a_i) \in R$, $a_i \in U$ if all the symbols in the left-hand side of rule r_i are present in the skin region. Thus the system never produces the right-hand side of a rule if not all of the symbols in the left-hand side of the rule are present in the skin region. This means that the set of facts the system derives

is always a subset of the set of facts that can be derived from A by Generalized Modus Ponens.

On the other hand, independently of the order in which the symbols in the left-hand side of the rule appear in the system, if all of the symbols in the left-hand side of the rule r_i are present in the skin region, the promoters of the rules ρ_{ij}, $1 \leq j \leq |V|$ guarantee that r_{i1} is transformed into a_i. This means that the system produces at least all facts that can be derived from A by Generalized Modus Ponens.

The conclusion is that the system always properly constructs the set of facts which can be derived from A by R.

If g can at all be derived from A, it is guaranteed to be produced at at most l-th step. At this point there will be t_l and g in the skin region and t_l will deterministically evolve into a YES. If, however, the symbol g is never produced, t_l will (correctly) evolve into a NO.

5 Backward Chaining

Backward chaining, along with forward chaining, is one of the two most commonly used methods of reasoning with inference rules. Backward chaining is also based on the modus ponens inference rule and is usually implemented by SLD resolution. Given a goal clause:

$$\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_i \vee \cdots \vee \neg L_n$$

with selected literal $\neg L_i$ and an input definite clause

$$L \vee (\neg K_1 \vee \neg K_2 \vee \cdots \vee \neg K_3)$$

in which the atom L unifies with the atom L_i, SLD resolution derives another goal clause, in which the selected literal is replaced by the negative literals of the input clause and the unifying substitution θ is applied:

$$\text{SUBST}(\theta, \neg L_1 \vee \neg L_2 \vee \cdots \vee (\neg K_1 \vee \neg K_2 \vee \cdots \vee K_n) \vee \cdots \vee \neg L_n)$$

As in the previous section, we will only consider zero-order logic in this section. In this case we may treat unification as equality. In this section we will take advantage of the possibility to only allow rules with exactly one symbol in the right-hand side.

In the case of zero-order logic, backward chaining is more complex than forward chaining, as are P systems doing backward chaining.

The description of the backward chaining algorithm is similar to depth-first search in a state space. In artificial intelligence backward chaining is often perceived in this way and a lot of considerations are built on top of this representation.

As usual, when implementing backward chaining in P systems, we would like to take as much advantage as possible of the parallelism offered by these devices.

The obvious way to exploit parallelism is exploring the branches of the deduction tree in parallel. More concretely, if, for a certain value of $Targets$, several rules $(V \to w) \in R$, $w \in Targets$ are found, the system should start investigating each of these branches in parallel. This approach is not equivalent to investigating all possible deduction branches in parallel.

Since we would like to explore a number of branches in parallel and since these branches are completely independent of one another, it would be natural to investigate each branch in a separate region. Because we would like to decide at each certain state how many new parallel explorations to start, membrane division would suit us greatly. This brings us to the conclusion that P systems with active membranes is what we need.

However, in P systems with active membranes one can only divide a membrane into two children membranes. A way to avoid this limitation would be to demand the set $R_w = \{V \to w \mid (V \to w) \in R\}$ to have no more than two elements. Any set of rules R can be transformed to satisfy this constraint by substituting every set of rules $R_w = \{V_1 \to w, V_2 \to w, \ldots, V_n \to w\}$, $n > 2$ with

$$\{V_1 \to z_1, V_2 \to z_1\} \cup \{z_{i-1} \to z_i, V_{i+1} \to z_i \mid 2 \leq i \leq n-2\} \cup \{z_{n-2} \to w, V_n \to w\}$$

The corresponding symbols z_i, $1 \leq i \leq n-2$ should be added to the new set of facts U'.

Note that this is not the most efficient transformation.

We introduce two morphisms $(')$ and $('')$ on a multiset $W = w_1 w_2 \ldots w_n$, $W \in U^+$:

$$W' = w_1' w_2' \ldots w_n'$$
$$W'' = w_1'' w_2'' \ldots w_n''$$

We also consider the corresponding specialization of these morphisms for sets.

Given that R satisfies the constraint specified above, consider the following P system with active membranes:

$$\Pi_b = (U_b, \{k, s\}, [\ [\ ]_k\ ]_s, w_k, w_s, R_b, s), \text{ where}$$

$$U_b = U \cup U' \cup U'' \cup \{\rho_i \mid (r_i : V_i \to w_i) \in R\} \cup \{f_0, f_1, f, c_0, c_1, c_2, c_3, c, l, p, q, \$_1, \$_2, \#\} \cup \cup\{t_i | 0 \le i \le 7\} \cup \{\text{YES}, \text{NO}\},$$

$$\begin{aligned} R_b = {} & \{[w'']_k^0 \to [\rho_i]_k^+ [\rho_j]_k^+ \mid \exists\{V_i \to w, V_j \to w\} \subseteq R, V_i \ne V_j\} \cup \\ & \cup\{[w'']_k^0 \to [\rho_i]_k^+ [\#]_k^+ \mid \exists(V_i \to w) \in R, \nexists(\alpha \to w) \in R, \alpha \ne V_i\} \cup \\ & \cup\{[qa \to a'a'']_k^0, [aa' \to a']_k^+, [aa \to a]_k^+ \mid a \in U\} \cup \\ & \cup\{[\rho_i \to qc_0 f_0 V_i]_k^+ \mid \exists(V_i \to \alpha) \in R, \alpha \subseteq U\} \cup \\ & \cup\{[f_0 \to f_1]_k^+, [l]_k^+ \to []_k^0 l, [l]_s^0 \to []_s^+ l, [l \to \lambda]_s^+\} \cup \\ & \cup\{[f_1 a \to a l]_k^+ \mid a \in U\} \cup \\ & \cup\{[c_{i-1} \to c_i]_k^+ \mid 1 \le i \le 3\} \cup \{[c_3 \to \lambda]_k^0, [c_3]_k^+ \to []_k^+ \$_0\} \cup \\ & \cup\{[\$_0 \to \$_1 \text{YES}]_s^+, [\$_0 \to \text{YES}]_s^0, [\text{YES NO} \to \text{YES}]_s^+, [\$_1]_s^0 \to []_s^+ \$_1\} \cup \\ & \cup\{[t_{i-1} \to t_i]_s^0 \mid 1 \le i \le 6\} \cup \\ & \cup\{[t_6 \to p t_7]_s^0, [t_7 \to t_2]_s^+, [t_7 \to \lambda]_s^0, [p \to \text{NO}]_s^0, [p]_s^+ \to []_s^0 p\}, \end{aligned}$$

$$w_k = qgA',$$

$$w_s = t_0.$$

This system decides, using backward chaining, whether g can be derived from A according to the rules in R. Handling cycles in inference rules ($\{a \to b, b \to a\}$) is a matter of further research.

Π_b works as follows. It starts with a single worker membrane with the label k, which contains the goal symbol g and A'. All primed symbols in the worker membrane are the symbols which have already been reduced once. The system will never reduce the same symbol twice in a worker membrane. The rule $[qa \to a'a'']_k^0$ marks g as reduced and creates a double primed copy of it. Double primed symbols are the symbols which are meant to be reduced.

The system includes two types of operations for reducing symbols: $[w'']_k^0 \to [\rho_i]_k^+ [\rho_j]_k^+$ and $[w'']_k^0 \to [\rho_i]_k^+ [\#]_k^+$. The first operation is done in the cases when $|R_w| = 2$. It creates two new worker membranes with polarization + for each of the two left-hand sides of the rules used for reduction. The second operation is performed when $|R_w| = 1$. This operation creates two worker membranes, but one of them contains # which will not allow the membrane to evolve further. It is important to realize that, even if $|R_w| = 1$, the corresponding P system rule need employ a membrane to avoid attempts to multiply apply rules.

In any of the newly created membranes the rule $[\rho_i \to qc_0 f_0 V_i]_k^+$ introduces the actual left-hand side of the corresponding inference rule, as well as several service symbols. In the next step the rule $[aa' \to a']_k^+$ removes any of the new symbols which have already been reduced. At the same time f_0 evolves into f_1 and c_0 into c_1. In the next step f_1 verifies whether not primed symbols are still present in the membrane. If there indeed are such symbols, an l is produced, which eventually re-polarizes the worker membrane to 0 and thus re-launching the

process of application of inference rules. The role of the symbol q is to assure that only one not yet reduced symbol is reduced.

In parallel with the rule $[aa' \to a']_k^+$, the rule $[aa \to a]_k^+$ is applied. It removes the duplicates which appear in situations when the worker membrane contains a and b and reduces b by the rule $V \to b$, $a \in V$. Note that the rule removing the already reduced symbols and the rule cannot be applicable to the same symbol at the same time, which removes concurrency effects.

If the worker membrane contains no more symbols which have not yet been reduced, Π_b concludes that we have discovered a resolution of g to the set of know atoms A. Since the rule $[f_1 a \to al]_k^+$ does not produce the symbol l, the symbols c_i evolve until c_3 ejects a $\$_0$ in the skin region. This symbol will eventually produce YES.

The skin membrane contains symbols t_i which check whether there still is some activity in the system. Each application of an inference rule (or two inference rules, when $|R_w| = 2$) takes 6 steps. At the very beginning, the symbol t_0 evolves into t_6. If $\exists(V \to g) \in R$, when t_6 is produced in the skin region, the skin region will also contain an l. t_6 evolves into pt_7 and, at the same time, l polarizes the skin membrane to $+$. This makes t_7 evolve into t_2, thus restarting the t_i loop, while p resets the skin polarization to 0.

If, however, no worker membrane produces an l any more, when p is produced in the skin, the skin polarization stays at 0. This forces p to produce a NO and erases t_7, thus breaking the t_i loop.

The symbol $\$_0$ will always appear in the skin region at the same time as pt_7. Two situations are possible: if there has just been at least one l in the skin, the skin will have polarization $+$. In this case t_2 is produced and p is erased. At the same time, $\$_0$ produces a $\$_1$ and YES. $\$_1$ polarizes the skin to $+$, thus stopping the t_i loop. In the other situation no instances of l are produced and, when pt_7 is produced in the skin, the polarization of this membrane is 0. In this case p will produce a NO, while t_7 will be erased, thus breaking the t_i loop. At the same time $\$_0$ will produce a YES, which will erase NO in the next step.

We remark that Π_b always stops, because any application of an inference rule necessarily leads to an eventual extension of the set of primed symbols within a worker membrane. This means that the time Π_b works in can be estimated as $O(|U|)$.

Π_b is a P system with active membranes with two polarizations and cooperative rules of type (a). It is unfourtunately necessary to use cooperation, too, in this context because we need to apply every rule only once. While it should be possible to implement backward chaining using purely non-cooperative rules, such approach would hurt the clarity of the solution and it is highly possible that the parallelism of P systems would not be fully used.

Π_b is notably more complicated than any of the P systems doing forward chaining. The reason for this situation is that we have only focused on zero-order logic so far, in which unification degenerates into equality and forward chaining becomes very straightforward to implement. In conventional programming, however, back-

ward chaining is sometimes preferred due to the fact that it is easier to satisfy memory restrictions.

6 Conclusion

In this paper we continued exploring the possibilities of solving reasoning problems with P systems, a topic which was started in [2]. The computing devices of P systems look appealing in this context due to two main reasons: the similarity of inference rules and multiset rewriting rules and the maximal parallelism. The similarity of the two types of rules allows a relatively natural transformation of reasoning problems into P systems and a rather efficient exploitation of the maximal parallelism.

We have only focused on zero-order logic in this paper, which resulted in quite simple P systems for forward chaining problems and more complicated devices for backward chaining. The difference in complexity appears because of the inherently recursive nature of backward chaining; and since one of our goals was to exploit the maximal parallelism, in the case of backward chaining we needed to branch off parallel explorations for each of the possibilities arousing at every reasoning step (after every SLD resolution).

This paper does not attempt to be exhaustive. One of the most evident questions is whether the P systems suggested in this paper can be optimized in the number of rules or control symbols. Another optimization criterion is the speed of the system. Although it is remarked in the paper that all P systems have the time complexity $O(|R|)$, Π_b is about five times slower on average than $\Pi_f^{(1)}$. In designing the P systems in this paper we tried to translate the reasoning tuple as intuitively as possible; maybe less intuitive transformations would operate in better time. A concrete question in this domain is whether it possible to design a general algorithm for constructing P systems which would solve reasoning problems in sublinear time (no matter which chaining algorithm is used).

A very important research question is how to handle the situations when R includes rules which form cycles. The presence of such cycles does not necessarily disrupt the functionality of the system, but may make it produce a falsely positive result.

Another relevant direction to explore is first-order logic. In zero-order logic we could comfortably translate facts to symbols and build relatively simple P systems. First-order logic poses serious questions, however, among which one of the most important ones is how to encode predicates in P systems and how to implement unification.

A problem we have only superficially talked of is universality. Because our focus was on intuitive transformations from a reasoning tuple to a P system, we didn't pay much attention to designing a P system which would solve all or a subset of reasoning problems using either chaining algorithm. This should be a relevant topic of theoretical research and could reveal further similarities between reasoning problems and certain kinds of P systems.

Acknowledgements

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

MAGN acknowledges the support of the projects TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence of the Junta de Andalucía, grant P08-TIC-04200.

References

1. Jago, M.: Formal Logic, *Humanities-Ebooks LLP*, 2007, ISBN 978-1-84760-041-7.
2. Gutiérrez-Naranjo, M. A., Rogozhin, V., Deductive databases and P systems, *Computer Science Journal of Moldova*, vol. 12, no. 1(34), 2004.
3. Apt, K. R.: *Logic Programming*, Handbook of Theoretical Computer Science. Elsevier Science Publishers B.V., 1990.
4. Bratko, I.: *PROLOG Programming for Artificial Intelligence*, Third Edition. Addison-Wesley, 2001.
5. Krishna S. N., Rama R.: A Variant of P Systems with Active Membranes: Solving NP-Complete Problems. *Romanian Journal of Information Science and Technology*, 2, 4 (1999), pp. 357-367.
6. Lloyd, J. W.: *Foundations of Logic Programming*, (2nd ed.) Springer, Berlin, 1987.
7. Păun, Gh., *Membrane Computing. An Introduction.* Springer-Verlag, 2002.
8. Păun, Gh., Rosenberg, G., Salomaa, A., Eds: *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2009.
9. The P systems web page. `http://ppage.psystems.eu/`

Towards Automated Verification of P Systems Using Spin

Raluca Lefticaru, Cristina Tudose, and Florentin Ipate

University of Pitesti, Department of Computer Science
Str. Targu din Vale 1, 110040, Pitesti, Romania
name.surname@upit.ro

Summary. This paper presents an approach to P systems verification using the Spin model checker. A tool which implements the proposed approach has been developed and can automatically transform P system specifications from P-Lingua into Promela, the language accepted by the well known model checker Spin. The properties expected for the P system are specified using some *patterns*, representing high level descriptions of frequently asked questions, formulated in *natural language*. These properties are automatically translated into LTL specifications for the Promela model and the Spin model checker is run against them. In case a counterexample is received, the Spin trace is decoded and expressed as a P system computation. The tool has been tested on a number of examples and the results obtained are presented in the paper.

1 Introduction

Membrane computing is a branch of natural computing, inspired from the structure and functioning of the living cell. Its models, called *P systems*, aim to simulate the evolution of a living cell, as well as the interaction or cooperation of cells in tissues, organs, or other types of populations of cells [16, 17]. P systems were introduced in 1998, in a seminal research report of Gheorghe Păun, further published as a journal paper [15].

The new field of membrane computing has known a fast development and many applications have been reported [2], especially in biology and bio-medicine, but also in unexpected directions, such as economics, approximate optimization and computer graphics [17]. Also, a large number of software tools for simulating P systems have been developed, many of them with the purpose of dealing with real world problems, such as those arisen from biology. An overview of the state of the art in P system software can be found in [17], chapter 17. The P-Lingua framework [9], one of the most promising software projects in membrane computing, proposes a new programming language, aiming to become a standard for the representation and simulation of P systems.

Designing a P system to solve a certain real world problem is a difficult task and many simulations are needed to check whether the proposed model behaves as expected. After designing a P system that aims to solve a given problem, a validation is needed, to ensure that the proposed model corresponds to what it is expected. One way to achieve this validation is to formally prove that the P system computations realize the given task. However, the formal proof is somehow hindered by the parallel and non-deterministic nature of the P systems. Consequently, automated tools, such as model checkers, would be very useful to prove or disprove 'on-the-fly' that the P system meets the expected specifications, expressed as temporal logic formulas.

Model checking is an automated technique for verifying if a model meets a given specification [4]. It has been applied for verifying models of hardware and software designs, such as sequential circuits designs, communication protocols, concurrent systems etc. A *model checker* is a tool that receives as input a property expressed as a temporal logic formula and a model of the system, given as an operational specification, and verifies, through the entire state space, whether the property holds or not. If a property violation is discovered then a counterexample is returned, that details why the model does not satisfy the property specified. Two widely used temporal specification languages in model checking are Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [3].

Spin is probably the most well-known LTL model checker [8]. It was written by Gerard Holzmann in the '80, developed over three decades at Bell Laboratories and it received in 2001 the prestigious ACM System Software Award. The transition systems accepted by SPIN (Simple Promela Interpreter) are described in the modelling language Promela (Process Meta Language) and the LTL formulas are checked using the algorithm advocated by Gerth et al. [7]. Spin can also operate as a simulator, following one possible execution path through the system and presenting the resulting execution trace to the user.

In this paper we present an approach to automatic translation of P systems into executable specifications in Promela, the language accepted by the Spin model checker, and its further verification using Spin. The paper intends to realize a bridge between P-Lingua, a very promising framework for defining and simulating P systems, and Spin, one of the most successful model checkers. The tool presented in the paper assists in designing and verifying P systems by automatically transforming the P-Lingua specifications into Promela. The properties expected for the P system are specified in a 'natural language', using an user-friendly interface, then they are automatically translated into LTL specifications for the Promela model; furthermore, the Spin model checker is run against them. In case a counterexample is received, the Spin trace is decoded and expressed in terms of a P system derivation.

Model checking based verification of P systems is a topic which has attracted a significant amount of research in the last years; the main tools used so far are Maude [1], Prism [18], NuSMV [13], Spin [12] and ProB [10]. This paper makes further advances in this area. Firstly, each model checker uses a particular language

for describing the models accepted. The activity of specifying a P system in a certain language, such as Promela for Spin, or SMV for NuSMV, can be tedious and error-prone. Many model checking tools cannot directly implement the transitions of a P system working in maximally parallel mode and consequently, the models obtained are complex, because they are simulating the parallelism using many sequential operations. With this respect, the tool presented here automatically transforms a P system definition file into an executable specification for the Spin model checker. The input file is the P-Lingua specification of a P system, which is an easy way of expressing the P systems and can also be used for simulation with the P-Lingua framework.

Secondly, the executable specifications written for different model checkers are not functionally equivalent with the P systems, for example they can contain extra states and variables corresponding to intermediate steps, which have no correspondence in the P system configurations [12]. For this reason, the P system properties that need to be verified, should be reformulated as properties of the executable implementation. The tool described in this paper takes the properties expressed in a natural language and transforms them into LTL formulas for the Promela model. It hides all the specialized information of the Spin model checker and provides the answer (true or false); in case a counterexample is found, this is decoded and expressed as a P system computation.

The paper is structured as follows. We start by describing the theoretical foundations of this approach in Section 2; the proposed framework is presented in Section 3. Some examples are explained in Section 4, the related work is presented in Section 5 and finally the conclusions are drawn in Section 6.

2 Background

2.1 P Systems

Before presenting our approach to P system verification, let us establish the notation used and define the class of cell-like P systems addressed in the paper. Basically, a P system is defined as a hierarchical arrangement of membranes, identifying corresponding regions of the system. Each region has an associated finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string $u \in V^*$, where the order is not considered, or by $\Psi_V(u)$. The following definition refers to cell-like P systems, with transformation and communication rules [16].

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

The membrane structure, μ, is denoted by a string of left and right brackets ($[_i$ and $]_i$), each with the label of the membrane i, it points to; μ also describes the position of each membrane in the hierarchy. The rules in each region have the form $u \rightarrow (a_1, t_1) \ldots (a_m, t_m)$, where u is a multiset of symbols from V, $a_i \in V$, $t_i \in \{in, out, here\}$, $1 \leq i \leq m$. When such a rule is applied to a multiset u in the current region, u is replaced by the symbols a_i with $t_i = here$; symbols a_i with $t_i = out$ are sent to the outer region or outside the system when the current region is the external compartment and symbols a_i with $t_i = in$ are sent into one of the regions contained in the current one, arbitrarily chosen. In the following definitions and examples when the target indication is *here*, the pair $(a_i, here)$ will be replaced by a_i. The rules are applied in maximally parallel mode which means that they are used in all the regions at the same time and in each region all the objects to which a rule *can* be applied *must* be the subject of a rule application [15].

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

We say that a rule is *cooperative* if it has at least two objects in its left hand side, e.g. $ab \rightarrow (c, in)(d, out)$. Otherwise, the rule is *non-cooperative*, e.g. $a \rightarrow (c, in)(d, out)$. Electrical charges, from the set $\{+; -; 0\}$, can be associated with membranes, as described in [16].

2.2 Linear Temporal Logic

The Linear Temporal Logic (LTL) was introduced by Amir Pnueli in 1977 [14] for the verification of computer programs. Compared to the branching time logic CTL (Computation Tree Logic) [4], LTL does not have an existential path quantifier (the **E** of CTL). An LTL formula has to be true over all paths, having the form $\mathbf{A}f$, where f is a path formula in which the only state subformulas permitted are atomic propositions. Given a set of atomic propositions AP, an LTL path formula [4] is either:

- If $p \in AP$, then p is a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\,\mathbf{U}\,g$ and $f\,\mathbf{R}\,g$ are path formulas, where:
  - The **X** operator ("neXt time", also written $\bigcirc$) requires that a property holds in the next state of the path.
  - The **F** operator ("eventually" or "in the future", also written $\Diamond$) is used to assert that a property will hold at some state on the path.
  - $\mathbf{G}f$ ("always" or "globally", also written $\Box$) specifies that a property, f, holds at every state on the path.

- $f\,\mathbf{U}\,g$ operator (**U** means "until") holds if there is a state on the path where g holds, and at every preceding state on the path, f holds. This operator requires that f has to hold *at least* until g, which holds at the current or a future position.
- $f\,\mathbf{R}\,g$ ("release") is the logical dual of the **U** operator. It requires that the second property holds along the path up to and including the first state where the first property holds. However, the first property is not required to hold eventually: if f never becomes true, g must remain true forever.

2.3 P System Specification in Promela

In this section we will present the theoretical background for verifying P systems using the Spin model checker, as proposed in [12].

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

In [11] it is explained how an associated Kripke structure can be built for a given P system. For this, the object multiplicities in the P systems membranes have to be restricted to a finite domain and so additional state variables and predicates are defined, following the guidelines from [6]. However, the Spin model checker cannot directly implement this kind of model and a Promela implementation is not functionally equivalent to the P system.

In the following we will present the transformation of a simple P system into a Promela model. For more details, [12] can be consulted and some examples can be downloaded from `http://fmi.upit.ro/evomt/psys/psys_spin.html`. Consider the one-membrane P system $\Pi = (V, \mu, w, R)$, with the alphabet $V = \{a_1, \ldots, a_k\}$ and the set of rules $R = \{r_1, \ldots r_m\}$ (each rule r_i has the form $u_i \rightarrow v_i, u_i, v_i \in V^*$). The Promela implementation of the P system will contain:

- k variables, labeled exactly like the objects from V, each one showing the number of occurrences of each object in the membrane, $a_i \in V$, $1 \leq i \leq k$;
- at most k auxiliary variables, labeled like the objects from the alphabet V plus a suffix p, each one showing the number of occurrences of each object a_i, produced in the current computation step;
- m variables n_i, $1 \leq i \leq m$, each one showing the number of applications of each rule $r_i \in R$, $1 \leq i \leq m$;
- one variable *state* showing the current state of the model, $state \in \{running, halt, crash\}$;
- one boolean variable $bStateInS$ expressing if the current configuration in the Promela model represents a state in the P system; $bStateInS$ is *false* when

intermediary steps are executed and *true* when the computation is over (a step in the P system derivation is completed); a corresponding atomic proposition `pInS` will evaluate whether $bStateInS$ holds;

- two constants, Max, the upper bound for the number of occurrences of each object $a_i \in V, 1 \leq i \leq k$, and Sup, the upper bound for the number of applications of each rule $r_i, 1 \leq i \leq m$;
- a set of propositions, which will be used in LTL formulas; they are introduced by `#define` and named suggestively. For example, `#define pn1 (n1>0)` is used to check if the rule r_1 has been applied at least once, `#define pa1 (a==1)` checks if the number of objects of type a is exactly 1.

In order to describe in Promela one computation step of a P system, a set of operations, additional variables and intermediary states are needed. For example, consider $\Pi_1 = (V_1, \mu_1, w_1, R_1)$, a simple one-membrane P system having $V_1 = \{s, a, b, c\}$, $\mu_1 = [_1]_1$, $w_1 = s$, $R_1 = \{r_1 : s \rightarrow ab; r_2 : a \rightarrow c; r_3 : b \rightarrow bc; r_4 : b \rightarrow c\}$. The following code excerpt corresponds to Π_1, when the current state is *running*:

```
bStateInS = false;
n1 = 0; n2 = 0; n3 = 0; n4 = 0;
ap = 0; bp = 0; cp = 0;
do
  :: s > 0 -> s = s - 1; n1 = n1 + 1; ap = ap + 1; bp = bp + 1
  :: a > 0 -> a = a - 1; n2 = n2 + 1; cp = cp + 1
  :: b > 0 -> b = b - 1; n3 = n3 + 1; bp = bp + 1; cp = cp + 1
  :: b > 0 -> b = b - 1; n4 = n4 + 1; cp = cp + 1
  :: else -> break
od;
a = a + ap; b = b + bp; c = c + cp;
if
  :: ( a > Max || b > Max || c > Max || s > Max ||
       n1 > Sup || n2 > Sup || n3 > Sup || n4 > Sup ) ->
            state = crash; bStateInS = true
  :: else ->
    if
      :: s == 0 && a == 0 && b == 0  ->
         state = halt; bStateInS = true
      :: else ->
          state = running; bStateInS = true
    fi
 fi
```

The `do-od` loop realizes the non-deterministic application of the rules. It is followed next by an `if` statement deciding the next state from the set $\{halt, crash, running\}$. The code is self-explanatory, for more details [12] can be consulted, so we will focus next on the properties to be checked.

Property	LTL specification
$\mathbf{G}\,p$	`[] (p || !pInS)`
$\mathbf{F}\,p$	`<> (p && pInS)`
$p\,\mathbf{U}\,q$	`(p || !pInS) U (q && pInS)`
$\mathbf{X}\,p$	`X (!pInS U (p && pInS))`
$p\,\mathbf{R}\,q$	`(p && pInS) V (q || !pInS)`
$\mathbf{G}\,(p \rightarrow q)$	`[] (!p || q || !pInS)`
$\mathbf{G}\,(p \rightarrow \mathbf{F}\,q)$	`[] ((p -> <>(q && pInS)) || !pInS)`

Table 1. Reformulating the P system properties for the Promela implementation

2.4 LTL Properties Transformation

The P system semantics is implemented for Spin as a sequence of transitions (or operations) and, consequently, additional intermediary states are introduced into the model. Furthermore, we consider that in the Promela executable model every possible path will contain infinitely often states corresponding to the P system configurations (i.e. the intermediary states do not form infinite loops). From these assumptions, it follows that every path in the P system has at least one corresponding path in the Promela model and vice versa. Furthermore, restrictions on the multiplicity of objects and rules applied are imposed.

The next step needed for model checking P systems with Spin is reformulating the properties to be verified in equivalent formulas for the associated Promela model. For example, a property like 'always $b > 0$' (the number of occurrences of b objects is always greater than 0) should become for the Promela model 'Globally $b > 0$ or not $pInS$' (we expect $b > 0$ only for configurations corresponding to the P system, but not for the intermediary steps).

In Table 1 we summarize the transformations of all LTL formulas for the Promela specification, as they are formally proved in [12].

3 Tool Description

The P system model-checking approach presented has been implemented in a software tool, which can perform 'on-the-fly' verification of properties expressed in a natural language. The tool, as well as the specifications of the P systems used in our tests, can be downloaded from the following web page: `http://fmi.upit.ro/evomt/psys/psys_spin.html`. An advantage of this tool is that the users do not need to be experts in formal verification or to write complex, specialized LTL formulas, in order to apply the model checking verification technique. They only need to specify the P-system, using P-Lingua, to choose the type of property to be checked and the conversion to Promela is performed automatically. If the property is false, the counterexample returned by Spin is parsed

by the tool and represented as a P system computation. A high-level overview of the process is presented in Fig. 1.

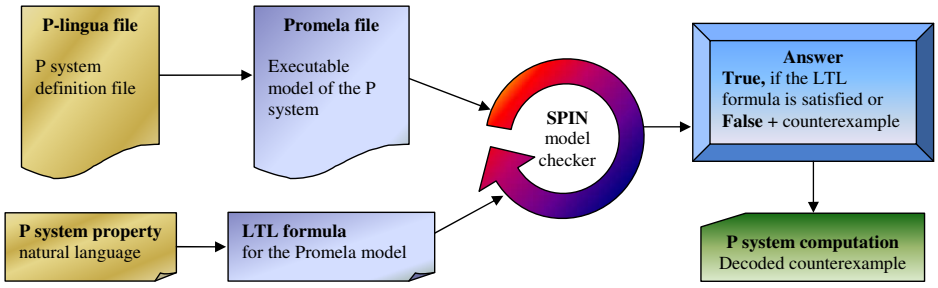


Fig. 1. Tool overview

In order to use the tool, the next steps are required:

- The user specifies the P-system in P-Lingua. He/she can also check, using the P-Lingua parsers, if the P system definition file is syntactically correct.
- The P-system is automatically transformed into a Promela specification.
- The user specifies some basic configuration properties over the system variables.
- The user selects a type of property to be verified, given in natural language, and the basic propositions involved in that property.
- The property is then automatically translated into an LTL formula suitable for the generated Promela model.
- The verification of the P system is performed automatically and if the answer returned by Spin is *false*, the tool will provide a counterexample expressed as a P system computation (with the configurations and set of rules applied at each step).

The main drawback of the application of previous model checking approaches to formal verification of P systems is the difficulty for non-expert users (P systems users) to formulate the appropriate properties in temporal logic. In our case, this could be further amplified by the fact that the LTL formulas would have to be transformed as described earlier. To alleviate this problem, we define some *patterns*, representing high level descriptions of frequently asked questions, formulated in *natural language*. This patterns, which simplify the specification of properties to be verified, are employed in our tool.

Let AP be a set of atomic propositions. An example of such proposition could be: *the number of b objects from a membrane m is greater than* 0. Let $p \in AP$. A basic configuration property ϕ over AP is defined as:

$$\phi ::= p \mid \phi \vee \phi \mid \phi \wedge \phi.$$

Each pattern is given as a natural language phrase that contains one or more basic configuration properties. The patterns considered so far are (in what follows

ϕ and ψ are basic configuration properties, the statement must hold on every computation):

- **Invariance (G ϕ):** a configuration in which ϕ is true must persist indefinitely.
- **Occurrence (F ϕ):** a configuration where ϕ is true will eventually occur.
- **Next occurrence (X ϕ):** a configuration where ϕ is true will occur after initial configuration.
- **Sequence (ϕ U ψ):** a configuration where ϕ is true is reachable and is necessarily preceded all the time by a configuration in which ψ is true.
- **Dual of sequence (ϕ R ψ):** on every computation, along the computation up to and including the first configuration where ϕ is true, in all the configurations ψ holds. However, a configuration where ϕ is true is not required to hold eventually.
- **Consequence (G ($\phi \rightarrow$ F ψ)):** if a configuration where ϕ is true occurs, then a configuration where ψ is true will eventually occur.
- **Instantly consequence (G ($\phi \rightarrow \psi$)):** if a configuration where ϕ is true occurs, then ψ holds also in that configuration.

These patterns can be used to formulate frequently asked questions, without worrying about their transformation into a temporal logic formula suited for our model. The transformation is performed automatically, employing the formulas from Table 1.

4 Case Studies

In this section, we present some examples of P system properties we have verified using the framework introduces previously. In order to simplify the presentation, we consider only one-membrane P systems. The first example is Π_1, the P system defined in Section 2. This is a simple P system, that has been used in several papers. In order to facilitate the comparison with other representations, such as SMV [11] or Event-B [10], Π_1 is presented among the examples on which we illustrate our approach. The other P systems have different properties, that can be verified (e.g. the number of c objects is always the square of b objects), or present polarizations.

4.1 Simple P Systems

Consider the following one-membrane P system: $\Pi_1 = (V_1, \mu_1, w_1, R_1)$, having $V_1 = \{s, a, b, c\}$, $\mu_1 = [_1]_1$, $w_1 = s$, $R_1 = \{r_1 : s \rightarrow ab; r_2 : a \rightarrow c; r_3 : b \rightarrow bc; r_4 : b \rightarrow c\}$. Its corresponding derivation tree is given in Fig. 4.1.

In order to realize the verification of the desired properties, the basic configuration propositions, which are part of the formulas must be specified first, for example:

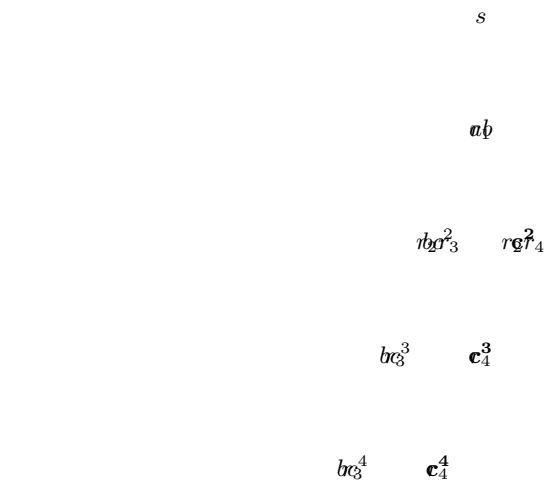


Fig. 2. Derivation tree for Π_1

Prop. ID	Property	Promela proposition
pab1	a==1 && b==1	#define pab1 (a==1 && b==1)
pab0	a==0 || b==0	#define pab0 (a==0 || b==0)
pc0	c==0	#define pc0 (c==0)
pa0c2	a==0 && c==2	#define pa0c2 (a==0 && c==2)
ps0	s==0	#define ps0 (s==0)
pa0	a==0	#define pa0 (a==0)
pa1	a==1	#define pa1 (a==1)
pc2	c==2	#define pc2 (c==2)
p1	s>0	#define p1 (s>0)
p2	c>1	#define p2 (c>1)

The first two columns are introduced by the user and they represent the basic configuration property and its name (or ID). The third column is based on the previous two and it is inserted automatically in the Promela specification. Having the basic proposition defined, more complex properties can be translated from a natural language into LTL formulas and verified using Spin. A set of properties checked for Π_1 is presented in Table 2.

A counterexample is received for every false property, e.g. $s \Longrightarrow ab$ corresponds to the second verified property from Table 2, which states that in the next configuration $a = 0 \wedge b = 0$. Similarly, the third property, expressing the invariance of c objects, $c = 0$, is falsified by $s \Longrightarrow ab \Longrightarrow bc^2$.

Consider the following one-membrane P system: $\Pi_2 = (V_2, \mu_2, w_2, R_2)$, having $V_2 = \{s, a, b, c, x\}$, $\mu_2 = [\,]_1$, $w_2 = s$, $R_2 = \{r_1 : s \to abcx; r_2 : a \to ab; r_3 : b \to bc^2; r_4 : x \to xc\}$. The computation of this P system is $s \Longrightarrow abcx \Longrightarrow ab^2c^4x \Longrightarrow ab^3c^9x \Longrightarrow ab^4c^{16}x \Longrightarrow \ldots$ and it does not halt. It can be easily observed that in every configuration the number of occurrences of c objects is the square of the

Properties	**LTL specifications for Spin**	**Truth**
Next occurrence: `pab1`	`X (!pInS U (pab1 && pInS))`	true
Next occurrence: `pab0`	`X (!pInS U (pab0 && pInS))`	false
Invariance: `pc0`	`[] (pc0 || !pInS)`	false
Occurrence: `pa0c2`	`<> (pa0c2 && pInS)`	true
Dual of sequence: `ps0, pc0`	`(ps0 && pInS) V (pc0 || !pInS)`	true
Sequence: `pc0, pa0`	`((pc0) || !pInS) U ((pa0) && pInS)`	true
Instantly consequence: `pb0, pa0`	`[](!pb0 || pa0 || !pInS)`	true
Consequence: `p1, p2`	`[]((p1 -> <>(p2 && pInS)) || !pInS)`	true

Table 2. Set of properties verified for Π_1.

number of b objects. Some properties, which take into account the value of the variable in the previous configuration, *var_old*, and the current computation *step*, are:

Prop. ID	Property	Promela proposition
`pbc`	`c==b*b`	`#define pbc (c==b*b)`
`p2`	`c-2*b_old-c_old-1==0`	`#define p2 (c-2*b_old-c_old-1==0)`
`pStep1`	`step>=1`	`#define pStep1 (step>=1)`
`pb`	`b==b_old+1`	`#define pb (b==b_old+1)`
`pc16`	`c==16`	`#define pc16 (c==16)`
`px0`	`x==0`	`#define px0 (x==0)`
`px1`	`x==1`	`#define px1 (x==1)`

Properties	**LTL specifications for Spin**	**Truth**
Invariance: `pbc`	`[] (pbc || !pInS)`	true
Occurrence: `pc16`	`<> (pc16 && pInS)`	true
Instantly consequence: `pStep1, pb`	`[] (!pStep1 || pb || !pInS)`	true
Instantly consequence: `pStep1, p2`	`[] (!pStep1 || p2 || !pInS)`	true
Sequence: `px0, px1`	`(px0 || !pInS) U (px1 && pInS)`	true

Table 3. Sample of properties verified for Π_2.

4.2 P Systems with Polarizations

Consider the following P system with charges: $\Pi_3 = (V_3, \mu_3, w_3, R_3)$, having $V_3 = \{a, b, c, d\}$, $\mu_3 = [\,]_1$, $w_3 = a^3$, $R_3 = \{r_1 : [a]_1^- \rightarrow [a, d]_1^0; r_2 : [a]_1^0 \rightarrow [ab]_1^+; r_3 : [a]_1^+ \rightarrow [ac]_1^-\}$. The computation of this P system is $[a^3]_1^0 \Longrightarrow [a^3b^3]_1^+ \Longrightarrow [a^3b^3c^3]_1^- \Longrightarrow [a^3b^3c^3d^3]_1^0 \Longrightarrow [a^3b^6c^3d^3]_1^+ \Longrightarrow [a^3b^6c^6d^3]_1^- \Longrightarrow \ldots$ and it does not halt. It can be easily observed that the number of each object is always a multiple

of 3; also, if the charge is 0, then the number of occurrences of b, c, d is equal. Examples of properties which can be formulated are:

Prop. ID	Property	Promela proposition
pa3	a%3==0	#define pa3 (a%3==0)
pagt0	a>0	#define pagt0 (a>0)
pba	b%a==0	#define pba (b%a==0)
pch0	ch==0	#define pch0 (ch==0)
pch1	ch==1	#define pch1 (ch==1)
pbcd	(b==c && c==d)	#define pbcd ((b==c && c==d))

Properties	**LTL specifications for Spin**	**Truth**
Invariance: pa3	[] ((pa3) || !pInS)	true
Instantly consequence pagt0, pba	[] (!pagt0 || pba || !pInS)	true
Instantly consequence ch0, pbcd	[] (!pch0 || pbcd || !pInS)	true
Instantly consequence ch1, pbcd	[] (!pch1 || pbcd || !pInS)	false

Table 4. Sample of properties verified for Π_2

5 Related Work

A first approach to P system model checking is presented in [1]. The authors use executable specifications written in Maude, a software system supporting rewriting and equational logic, to verify LTL properties of P systems.

The decidability of model-checking properties for P systems has been analysed in [5, 6] and the experiments realized show that Spin is preferable over Omega 'to serve as the back-end solver in a future P system model-checker'.

The probabilistic model checker Prism is employed in [18] to answer specific questions about stochastic P systems.

An approach to P system test generation, based on model checking, is presented in [11, 13] and uses the NuSMV symbolic model checker. This approach is compared with P system model checking using Spin in [12] and the experimental results obtained show that Spin achieves better performance with P system models. A very recent work on P system verification [10] uses the ProB model checker to verify P systems represented in Event-B, a modelling language considered to be an evolution of the B language.

6 Conclusions and Future Work

In this paper, we present a method to automatically verify P systems using the Spin model checker. The theoretical foundations of this approach have been presented

in [12] and its advantages have been shown, in comparison to previous work, that use another main stream model checker, NuSMV [13].

The tool presented in this paper is intended to help designing and verifying P systems by automatically transforming the P-Lingua specifications into Promela, the language accepted by the Spin model checker. The P system properties are specified in a natural language after which they are translated automatically into LTL specifications for the Promela model and then the Spin model checker is run against them. In case a counterexample is received, the Spin trace is decoded and expressed as a P system computation.

Future work consists in extending the tool to accept other classes of P systems, with division and dissolving rules. More experiments will be performed to determine the performance of the Spin model checker for more complex systems, such as those solving SAT problems.

Acknowledgment

This work was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 643/2008.

References

1. Oana Andrei, Gabriel Ciobanu, and Dorel Lucanu. Executable specifications of P systems. In Giancarlo Mauri, Gheorghe Păun, Mario Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 3365 of *Lecture Notes in Computer Science*, pages 126–145. Springer Berlin / Heidelberg, 2005.
2. Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Păun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.
3. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
4. Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
5. Zhe Dang, Oscar Ibarra, Cheng Li, and Gaoyan Xie. On model-checking of P systems. In Cristian Calude, Michael Dinneen, Gheorghe Păun, Mario Pérez-Jímenez, and Grzegorz Rozenberg, editors, *Unconventional Computation*, volume 3699 of *Lecture Notes in Computer Science*, pages 82–93. Springer Berlin / Heidelberg, 2005.
6. Zhe Dang, Oscar H. Ibarra, Cheng Li, and Gaoyan Xie. On the decidability of model-checking for P systems. *Journal of Automata, Languages and Combinatorics*, 11(3):279–298, 2006.
7. Rob Gerth, Doron Peled, Moshe Y. Vardi, R. Gerth, Den Dolech Eindhoven, D. Peled, M. Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *In Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
8. Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.
9. http://www.p lingua.org. The P-lingua website. last visited, April 2011.

10. Florentin Ipate and Adrian Ţurcanu. Modelling, verification and testing of P systems using Rodin and ProB. In *Ninth Brainstorming Week on Membrane Computing (BWMC 2011)*, page this volume, 2011.
11. Florentin Ipate, Marian Gheorghe, and Raluca Lefticaru. Test generation from P systems using model checking. *Journal of Logic and Algebraic Programming*, 79(6):350–362, 2010.
12. Florentin Ipate, Raluca Lefticaru, and Cristina Tudose. Formal verification of P systems using Spin. *International Journal of Foundations of Computer Science*, 22(1):133–142, 2011.
13. Raluca Lefticaru, Florentin Ipate, and Marian Gheorghe. Model checking based test generation from P systems using P-lingua. *Romanian Journal of Information Science and Technology*, 13(2):153–168, 2010. Special issue on membrane computing, devoted to Eighth Brainstorming Week on Membrane Computing (selected and revised papers).
14. Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
15. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
16. Gheorghe Păun. *Membrane Computing: An Introduction.* Springer-Verlag, 2002.
17. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
18. Francisco José Romero-Campero, Marian Gheorghe, Luca Bianco, Dario Pescini, Mario J. Pérez-Jiménez, and Rodica Ceterchi. Towards probabilistic model checking on P systems using PRISM. In Hendrik Jan Hoogeboom, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing - 7th International Workshop, WMC 2006, Revised, Selected, and Invited Papers*, volume 4361 of *Lecture Notes in Computer Science*, pages 477–495. Springer, 2006.

MP Modeling of Glucose-Insulin Interactions in the Intravenous Glucose Tolerance Test

Vincenzo Manca, Luca Marchetti, and Roberto Pagliarini

University of Verona
Department of Computer Science
Strada Le Grazie 15, 37134 Verona, Italy
{vincenzo.manca,luca.marchetti,roberto.pagliarini}@univr.it

Summary. The Intra Venous Glucose Tolerance Test (IVGTT) is an experimental procedure in which a challenge bolus of glucose is administered intra-venously and plasma glucose and insulin concentrations are then frequently sampled. An open problem is to construct a model representing simultaneously the entire control system. In the last three decades, several models appeared in the literature. One of the mostly used one is known as the *minimal model*, which has been challenged by the *dynamical model*. However, both the models have not escape from criticisms and drawbacks. In this paper we apply Metabolic P systems theory for developing new physiologically based models of the glucose-insulin system which can be applied to the Intra Venous Glucose Tolerance Test. We considered ten data-sets obtained from literature and for each of them we found an MP model which fits the data and explains the regulations of the dynamics. Finally, further analysis are planned in order to define common patterns which explain, in general, the action of the glucose-insulin control system.

1 Introduction

Glucose is the primary source of energy for body's cells. It is transported from the intestines or liver to body cells via the bloodstream, and is absorbed by the cells with the intervention of the hormone *insulin* produced by the pancreas. Blood glucose concentration is a function of the rate of glucose which enters the bloodstream, the glucose appearance, balanced by the rate of glucose which is removed from the circulation, the glucose disappearance. Normally, in mammals this concentration is tightly regulated as a part of metabolic homeostasis. Indeed, although several exogenous factors, like food intake and physical exercise, affect the blood glucose concentration level, the pancreatic endocrine hormones insulin and glucagon[1] keep this level in the range $70-110$ mg/dl. When the blood glucose concentration level is high, the pancreatic $\beta-$cells release insulin which lowers that concentration by

[1] Others gluco-regulatory hormones are: amylin, GLP-1, glucose-dependent insulinotropic peptide, epinephrine, cortisol, and growth hormone.

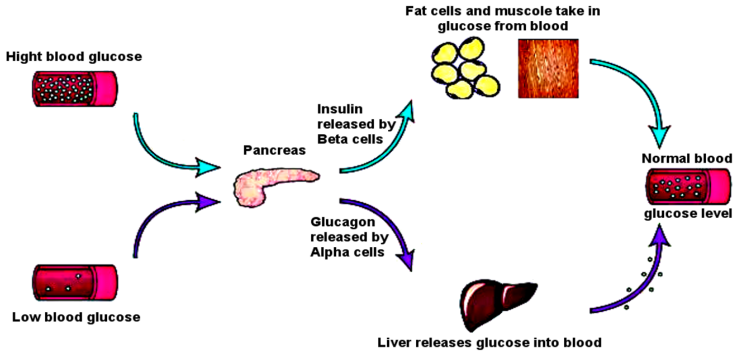


Fig. 1. The glucose homeostasis.

inducing the uptake of the excess glucose by the liver and other cells and by inhibiting hepatic glucose production. On the contrary, when the glucose level is low, the pancreatic $\alpha-$cells release glucagon that results in increasing the blood glucose level by acting on liver cells and causing them to release glucose into the blood[2] (see Figure 1).

If the plasma glucose concentration level is constantly out of the usual range, then we are in presence of blood glucose problems. In particular, when this level is constantly higher than the range upper bound (which is referred to as *hyperglycemia*), we are in presence of *Diabetes*: a dreadfully severe and pervasive illness which concerns a good number of structures in the body. Diabetes is classified into two main categories known as *type I* and *type II*, respectively. Type I diabetes is an illness concerning the pancreas during which the body demolishes its individual β-cells and the pancreas is no longer capable of making insulin. By means of no insulin to stir glucose within the body units, glucose assembles in the bloodstream and the concentrations rise. This category is most widespread among citizens below 30 and frequently appears in early days. The crest beginning is 12-14 years of period. Insulin injections are necessary for the residue of the victims' life. Luckily, Type I Diabetes results in $5-10\%$ of all categories of diabetes [30]. In quick disparity, Type II diabetes asserts the remaining 90%. It typically begins at the age of 35 or older and is particularly widespread in the aged. This type of diabetes may include an amalgamation of troubles. The pancreas is at rest capable to compose insulin, however regularly it does not compose sufficient or/and the units are not capable to utilize the insulin. Contrasting type I diabetes, insulin injections are not at all times essential, since the body is capable of at rest making a little insulin. Every now and then oral prescriptions, habitual work outs and high-quality nourishment are capable to controlling the elevated glucose heights. However, in both the types of diabetes, the illness can lead to several complications like retinopathy, nephropathy, peripheral neuropathy and blindness [6].

[2] We refer the reader to [24] for a deeper description of the processes that underlies the glucose-insulin system.

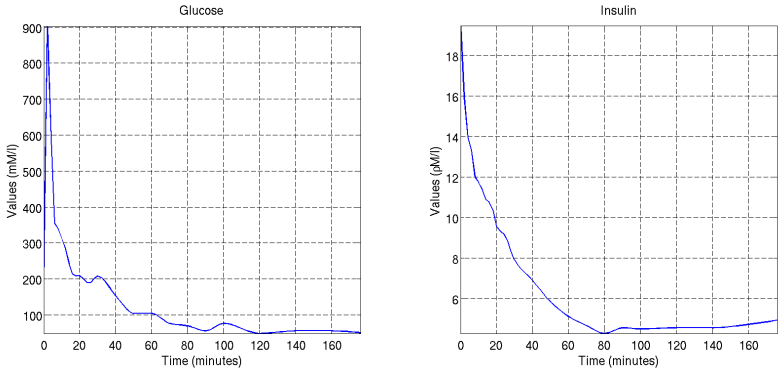


Fig. 2. Plots of a IVGTT data-set starting from the time of the glucose injection. The glucose dynamics is given on the left, while the insulin dynamics is given on the right.

While different regulatory interactions in the pathogenesis of this disease remain to be clarified [9] the number of diabetic patients is increasing [33]. This motivates researches to study the glucose-insulin endocrine regulatory system. In particular, the glucose-insulin system has been the object of repeated, mathematical modelling attempts. The majority of the proposed models were devoted to the study of the glucose-insulin dynamics by considering experimental data obtained by *intravenous glucose tolerance test*, shortly *IVGTT*, and the *oral glucose tolerance test*, shortly *OGTT*. In these models, the insulin-glucose system is assumed to be composed of two linked subsystems modelling the insulin action and the glucose kinetics, respectively. Since the action of insulin is delayed with respect to plasma glucose, the subsystems of insulin action typically includes a delay.

However, considering the limits of the existing mathematical models, a need exists to have reliable mathematical models representing the glucose-insulin system. The mere fact that several models have been proposed [4, 14, 23] shows that mathematical and physiological considerations have to be carefully integrated when attempting to represent the glucose-insulin regulatory mechanism. In particular, in order to model the IGVTT, a reasonably simple model is required. It has to have a few parameters to be estimated and has to have dynamics consistent with physiology and experimental data. Further, the model formulation, while applicable to model the IGVTT, should be logically and easily extensible to model other envisaged experimental procedures.

2 The intravenous glucose tolerance test

The *intravenous glucose tolerance test* focuses on the metabolism of glucose in a period of 3 hours starting from the infusion of a bolus of glucose at time $t = 0$. It is based on the assumption that, in a healthy person, the glucose concentration decreases exponentially with time following the loading dose (see Figure 2). It

has been recommended as a method to asses the use of insulin in order to identify subjects which may be diabetics [26]. This test makes use of an interaction between clearance of insulin from β-cells and the actions of insulin to accelerate glucose disappearance and to inhibit endogenous glucose production.

The IVGTT starts by rapidly, less that 3 minutes, inject into the blood stream of a subject a 33% glucose solution (i.e. $0.33g/Kg$) in order to induce an impulsive increase of the plasma concentrations of glucose and insulin. These concentrations are measured, by taking blood samples, during a period of three hours beginning at injection. The samples are then analysed for glucose and insulin context. In fact, in a healthy person, after this time interval the glucose and insulin plasma concentrations return normal (i.e. they return to their basal levels). Differently, this does not happen in a sick person.

Qualitatively, the plasma glucose level starts at a peak due to the injection, drops to a minimum which is below the basal glucose level, and then gradually returns to the basal level. At the same time, the plasma insulin concentration rapidly rises to a peak which follows the injection, drops to a lower level which is still above the basal insulin level, rises again to a lesser peak, and then gradually drops to the basal level. Depending to the state of the patient, there can be wide variations from this response. The glucose concentration may not drop below the basal level, the first peak of insulin level may have different amplitude, there may be no secondary peak in insulin concentration, or there may be more than two peaks in insulin.

3 Mathematical models of the intravenous glucose tolerance test

A variety of mathematical models, statistical methods and algorithms have been proposed to understand different aspects of diabetes. In this section we briefly review the two mathematical models which had the most important impact in diabetology for modelling the intravenous glucose tolerance test. They have been useful to assess physiological parameters and to study the glucose-insulin interactions. However, they have not escape from criticism and drawbacks.

Although several others models have been proposed [2], the real start of modeling glucose-insulin dynamics is due to the *minimal model* developed in [3, 32]. It has been characterized as the simplest model which is able to describe the glucose metabolism reasonably well by using the smallest set of identifiable and meaningful parameters [3, 27]. Several versions based on the minimal model have been proposed, and the reader can find further information on them in [2, 7]. The minimal model has been formulated by using the following system of differential equations:

$$\begin{aligned}\frac{dG(t)}{dt} &= -\left(p_1 + X(t)\right) G(t) + p_1 G_b \\ \frac{dX(t)}{dt} &= -p_2 X(t) + p_3 \left(I(t) - I_b\right) \\ \frac{dI(t)}{dt} &= p_4 \left(G(t) - p_5\right) t - p_6 \left(I(t) - I_b\right)\end{aligned} \tag{1}$$

where $G(t)$ $[mg/dl]$ and $I(t)$ $[\mu UI/ml]$ are plasma glucose and insulin concentration at time t $[min]$, respectively. $X(t)$ $[min^{-1}]$ is an auxiliary function which models the time delay of the insulin consumption on glucose. G_b and I_b are the subject baseline blood glucose and insulin concentration, while p_i, for $i = 1, 2, \ldots, 6$, are the model's parameters (we refer the reader to [3, 32] for all the details connected to these parameters). The first two equations of (1) represent the glucose disappearance subsystem, while the third one describes the insulin kinetic subsystem. In the second subsystem, the following rule is applied:

$$\left(G(t) - p_5\right) = \begin{cases} \left(G(t) - p_5\right) & \text{if } \; G(t) > p_5 \\ 0 & \text{if } \; G(t) \leq p_5 \end{cases} \tag{2}$$

while the multiplication by t is introduced to approximate the hypothesis that the effect of circulating hyperglicemia on the rate of pancreatic secretion of insulin is proportional both to the attained hyperglicemia and to the time delay from the glucose injection [32].

Although (1) is very useful in physiology research, it has some dynamical and mathematical drawbacks. First, some results produced by this model are not realistic [10]. Second, the glucose-insulin regulatory mechanism is an integrated dynamical system having feedback regulations, while the minimal model is composed of two subsystems. The parameters of these two subsystems are to be separately fitted from the available data, but by following this approach an internal coherency check is omitted. Last, the artificial non-observable variable $X(t)$ is introduced to model the delay in the action of insulin.

To overcome these drawbacks the *dynamical model* has been proposed in [10]:

$$\begin{aligned}\frac{dG(t)}{dt} &= -b_1 G(t) - b_4 I(t) G(t) + b_7 \\ G(t) &\equiv G_b \quad \forall t \in [-b_5, 0) \\ \frac{dI(t)}{dt} &= -b_2 I(t) + \frac{b_6}{b_5} \int_{t-b_5}^{t} G(s) ds.\end{aligned} \tag{3}$$

It is a delay integro-differential equation model which is a more realistic representation of the glucose-insulin dynamics which follows an IVGTT. Although it retains the physiological hypotheses underlying the first equation of (1), non-observable state variables are not introduced. Moreover, the physiological assumption underlying the third equation of (1), that pancreas is able to linearly increase its rate of insulin production with respect to the time, is not taken into account. The dynamical model assumes that the glucose concentration depend *i)* on insulin-independent

net glucose tissue uptake, *ii)* on spontaneous disappearance and *iii)* on constant liver glucose production. The insulin concentration, instead, is assumed to depend *i)* on a spontaneous constant-rate decay, which is due to the insulin catabolism, and *ii)* on pancreatic secretion. In particular, the insulin secretion at time t is assumed to be proportional to the average value in the b_5 minutes which precede t, where b_5 is assumed to lie in a range from 5 to 30.

The term $\frac{b_6}{b_5}\int_{t-b_5}^{t} G(s)ds$ represents the *decaying memory kernel* [8], which is introduced to model the time delay. The physiologic meaning of the delay kernel reflects the pancreas' sensitivity to the blood glucose concentration. At a given time t, the pancreas will produce insulin at a rate proportional to the suitably weighted average of the plasma glucose concentrations in the past.

The dynamical model allows simultaneous estimation of both insulin secretion and glucose uptake parameters. However, it is conceivable that the dynamical model may not be considerable appropriate under all circumstance [25]. This is due to the fact that the IVGTT data related to several subjects could be best fitted by using different delay kernels. Therefore, an extension of (3) is proposed in [25], where a generic weight function ω is introduced in the delay integral kernel modeling the pancreatic response to glucose level. In this way, the second equation of (3) becomes:

$$\frac{dI(t)}{dt} = -b_2 I(t) + b_6 \int_0^{\infty} \omega(s) G(t-s) ds \tag{4}$$

where $\omega(s)$ is assumed to be a non-negative square integrable function on $\mathbb{R}^+ = [0, \infty)$, such that $\int_0^{\infty} \omega(s)ds = 1$ and $\int_0^{\infty} s \cdot \omega(s)ds$ is equal to the average time delay. The idea is that different patients populations show different shapes of the kernel function ω, and then suitable parametrization of such a function could offer the possibility to differentiate between patient populations by means of experimental parameter identification.

Despite the models (3) and (4) solve the drawbacks of the minimal model, they made some assumptions that may not be realistic. The main restriction regards the way used to introduce the delay, for which the justification is only based on a subjective assumption. This limit implies the study of others ways to consider the time delay. To this end, an alternative approach to incorporate the time delay is analyzed in [13], where the authors propose a model which includes (3) and (4) as special cases. In this model, the delay is modelled by using a Michaelis-Menten form, and the effective secretion of insulin at time t is assumed to be regulated by the concentrations of glucose in the b_5 minutes which precede time t instead of the average amount in that period.

4 MP modelling

An important problem of systems biology is the mathematical definition of a dynamical system which explains the observed behaviour of a phenomenon by increasing what is already known about it. An important line of research of biological modelling is aimed at defining new classes of discrete models avoiding some

limitations of classical continuous models based on ordinary differential equations (ODEs). In fact, very often, the evaluation of the kinetic reaction rates is problematic because it may require measurements hardly accessible in living organisms. Moreover, these measurements dramatically alter the context of the investigated processes. In contrast to ODEs, *Metabolic P systems* (MP systems) [18, 16, 17, 15], based on Păun's P systems [28], were introduced for modelling *metabolic systems*.

In MP systems no single instantaneous kinetics are addressed, but rather the variation of the whole system under investigation is considered, at discrete time points, separated by a specified macroscopic interval τ. The dynamics is given along a sequence of steps and, at each step, it is governed by partitioning the matter among reactions which transform it. Metabolic P systems proved to be promising in many contexts and their applicability was tested in many situations where differential models are prohibitive due to the unavailability or the unreliability of the kinetic rates [15, 21, 19, 20, 22, 5].

A Metabolic P system is essentially a multiset grammar where multiset transformations are regulated by functions. Namely, a rule like $a + b \to c$ means that a number u of molecules of kind a and u of kind b are replaced by u molecules of type c. The value of u is the *flux* of the rule application. Assume to consider a system at some time steps $i = 0, 1, 2, \ldots, t$, and consider a substance x that is produced by rules r_1, r_3 and is consumed by rule r_2. If $u_1[i], u_2[i], u_3[i]$ are the fluxes of the rules r_1, r_2, r_3 respectively, in the passage from step i to step $i + 1$, then the variation of substance x is given by:

$$x[i + 1] - x[i] = u_1[i] - u_2[i] + u_3[i].$$

In an MP system it is assumed that in any state the flux of each rule is provided by a function, called *regulator*. Substances, reactions, and regulators (plus parameters which are variables different from substances occurring as arguments of regulators) specify a discrete dynamics at steps indexed in the set $\mathbb{N}$ of natural numbers. Moreover, a *temporal interval* τ, a conventional *mole size* ν, and substances masses are considered, which specify the time and population (discrete) granularities respectively. They are *scale factors* that do not enter directly in the definition of the dynamics of a system, but are essential for interpreting it at a specific physical level of mass and time granularity.

Here we apply an algorithm, called *Log-Gain Stoichiometric Stepwise Regression* (LGSS) [19], to define new MP models which describe the glucose-insulin dynamics in the IVGTT. LGSS represents the most recent solution, in terms of MP systems, of the inverse dynamics problem, that is, of the identification of (discrete) mathematical models exhibiting an observed dynamics and satisfying all the constraints required by the specific knowledge about the modelled phenomenon. The LGSS algorithm combines and extends the log-gain principles developed in the MP system theory [17, 15] with the classical method of Stepwise Regression [12], which is a statistical regression technique based on Least Squares Approximation and a statistical F-test [11]. The method can be correctly applied independently

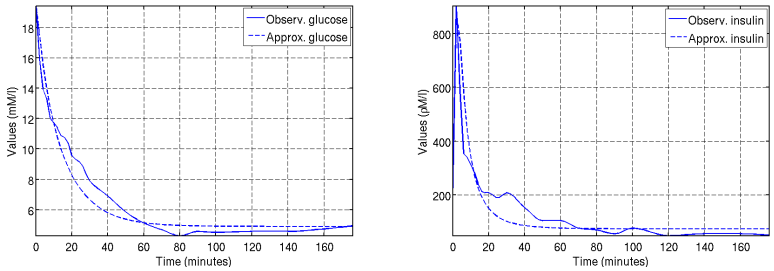


Fig. 3. The dynamics calculated by means of the MP grammar given in Table 1.

from any knowledge about reaction rate kinetics, and can provide, with respect to differential models, different and even simpler mathematical formulations.

The first MP grammar we give is the one of Table 1 which models the dynamics depicted in Figure 2. The model is given by 2 substances (G for the blood glucose level and I for the level of insulin) and 4 rules, the first two related to glucose and the others related to insulin: *i)* r_1: constant release of glucose in the blood, *ii)* r_2: glucose disappearance due to a term which represents the normal decay of glucose (depending on G) and to a term which indicate the action of insulin (depending on both G and I), *iii)* r_3: release of insulin by the pancreas which depends on the blood glucose level, and *iv)* r_4: normal decay of insulin.

The MP grammar is defined for a value of τ of two minutes[3] (which gives the length of the time interval between two consecutive computed step) and allows the calculation of the curves depicted in Figure 3. The dynamics is quite close to the data-set we started from. In fact, the multiple coefficients of determination R^2_G and R^2_I, calculated to estimate the goodness of the approximation for glucose and insulin [1], are equal to 0.94 and 0.87 respectively[4]. The usage of the term G^3 in φ_3, against the possibility of choosing monomials of G with lower degree, expresses the high sensitivity of the pancreas $\beta-$cells for the blood glucose level when they release insulin.

[3] In order to maintain the models as accurate as possible, we adopt here a time unit τ of two minutes because it is the minimal time granularity used in the data-sets we considered.

[4] The coefficient value ranges from 1, when the regression model perfectly fits the data, to 0 according to the goodness of the model fit.

$r_1 : \emptyset \to G$	$\varphi_1 = 0.6$
$r_2 : G \to \emptyset$	$\varphi_2 = 0.12G + 1.6 \cdot 10^{-6} G^2 I$
$r_3 : \emptyset \to I$	$\varphi_3 = 49.9 + 0.1G^3$
$r_4 : I \to \emptyset$	$\varphi_4 = 0.84I$

Table 1. The MP grammar which models the dynamics given in Figure 2 ($\tau = 2$ min).

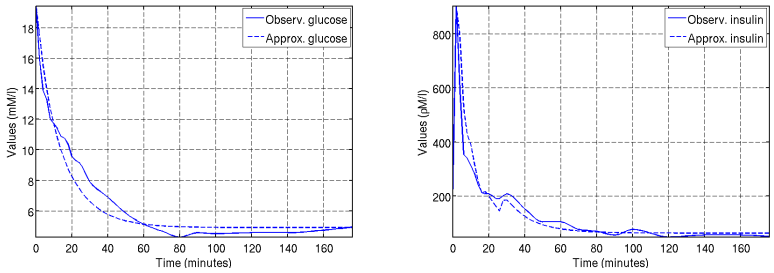


Fig. 4. The dynamics calculated by means of the MP grammar given in Table 2.

The formula of each regulator has been calculated by means of LGSS which selects suitable linear combinations starting from a given set of possible basic functions, called regressors, associated to each rule. Due to the biological meaning given to each reaction, in our analysis we forced: *i)* φ_1 to be a constant, *ii)* φ_2 to be a linear combination of monomials of G and I, *iii)* φ_3 to be a linear combination of monomials of G, and *iv)* φ_4 to depend on I. These assumptions, however, do not take into account the time delays which occur in the insulin release reducing the precision of the models. If we consider the dynamics of Figure 3, for example, the simulation fails to describe the insulin peak which occurs between the 20th and the 40th minute. This missing peak is quite small and for this reason our approximation seems to be enough precise, but if we try to define new MP grammars for other data-sets related to the IVGTT, we reach very soon situations in which the missing peaks are very high causing a dramatical lost of precision.

In the differential models introduced in Section 3, the delay of the insulin release is approached by adding artificial substances or by considering a delay integral kernel. Here, instead, we solve the problem by assuming that φ_3 is given by a linear combinations of monomial of G and of its memories. This permits to point out in a more natural and detailed way the different delays which act in the insulin production. If we indicate by $G^t = (G[i]|0 \leq i \leq t)$ the vector containing the time-series of glucose in a given data-set, we define the time-series G^t_{-m} related to the memory of glucose shifted m steps after as the vector

$$G^t_{-m} = (\underbrace{G_b, G_b, \ldots, G_b}_{m \ \ times}, G[0], G[1], \ldots, G[t-m])$$

where G_b is the basal value of the blood glucose level[5]. Memories are very simple to be managed in MP systems and increase a lot the approximation power of the models as showed in [21], where memories have been applied in the context of periodical function approximation.

The extension of the MP grammar of Table 1 which considers glucose memories is given in Table 2, while the new calculated dynamics is depicted in Figure 4.

[5] Since during the IVGTT the glucose level gradually returns to its basal level, here we assume G_b to be equal to the last value of the considered glucose time-series.

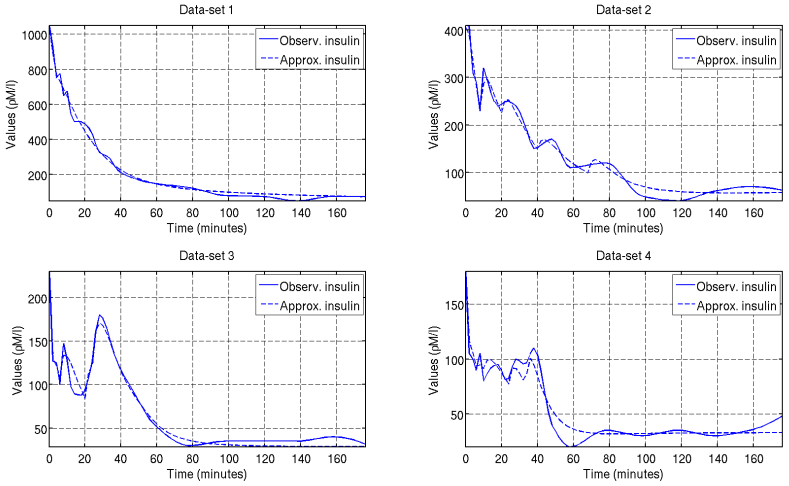
The new model provides a better data fitting for the insulin curve. The multiple coefficient of determination for the insulin is increased from 0.87 to 0.95. Moreover φ_3 gives now an idea of the different phases which act in the blood release of insulin by pointing out their strength (given by the degree of the selected monomials) and their delay (given by the delay of the selected memories).

In our analysis we considered ten different data-sets published in literature and obtained by applying the intravenous glucose tolerance test to ten healthy patients. All subjects have negative family histories for diabetes and other endocrine diseases. During the test, the patients were on no medications and had no current illness. Each test has been performed during the morning after an overnight fast, and for the three days preceding the test each subject followed a diet composed of 55% carbohydrates, 30% fats, and 15% proteins. The curves of the considered data-sets are very different form each other, especially the curve related to the insulin dynamics which exhibits values and peaks of different height and at different delays. In all the cases, however, we found MP models which provide good data fitting (the average of the calculated multiple coefficients of determination for all the models is greater than 0.95 for both glucose and insulin). In Table 3 we provide the regulators related to four of the considered data-sets, and the plotting of the corresponding calculated dynamics for the insulin. The depicted dynamics exhibit examples of all the different scenarios we observed concerning the insulin release in our data-sets. We can have situations where the insulin curve exhibits many peaks which model the different release phases, or we can have dynamics without significant peaks but that are in any case modelled by a delayed insulin secretion (this is the case of data-set 1).

The total number of monomials used to define φ_3 can be changed by acting on the thresholds used by LGSS during the computing of its statistical tests. The models provided here have been defined trying to balance their simplicity with their power of approximation. Each model provides a sort of picture of the metabolism of the subject which have been analysed.

$r_1 : \emptyset \to G$	$\varphi_1 = 0.6$
$r_2 : G \to \emptyset$	$\varphi_2 = 0.12G + 1.6 \cdot 10^{-6} G^2 I$
$r_3 : \emptyset \to I$	$\varphi_3 = 1.5 \cdot 10^{-5} G^6 + 0.25 G^2_{-6} + 0.17 G^2_{-8}$ $+2.65 G_{-16} + 3.6 G_{-26}$
$r_4 : I \to \emptyset$	$\varphi_4 = 0.65 I$

Table 2. The MP grammar which models the dynamics given in Figure 2 ($\tau = 2$ min) enriched with the usage of glucose memories (subscripts give the delay in minutes of each memory).



Data-set	Regulators
1	$\varphi_1 = 0.011$ $\varphi_2 = 6.6 \cdot 10^{-5} GI$ $\varphi_3 = 0.5 G^2_{-4}$ $\varphi_4 = 0.16 I$
2	$\varphi_1 = 0.056$ $\varphi_2 = 5.2 \cdot 10^{-4} I + 8.1 \cdot 10^{-5} GI$ $\varphi_3 = 3.76 \cdot 10^{-6} G^7 + 0.74 G^2_{-8} + 0.02 G^3_{-20} + 0.21 G^2_{-40} + 10^{-4} G^5_{-68}$ $\varphi_4 = 0.49 I$
3	$\varphi_1 = 0.12$ $\varphi_2 = 0.02 G + 1.9 \cdot 10^{-4} GI$ $\varphi_3 = 0.04 G^3_{-2} + 3.3 \cdot 10^{-5} G^6_{-6} + 0.44 G^2_{-20} + 0.04 G^3_{-24}$ $\varphi_4 = 0.5 I$
4	$\varphi_1 = 0.11$ $\varphi_2 = 6.2 \cdot 10^{-4} GI$ $\varphi_3 = 0.1 G^2_{-2} + 0.9 G_{-6} + 1.07 G_{-10} + 2.4 \cdot 10^{-4} G^4_{-24}$ $+5.4 \cdot 10^{-7} G^6_{-32} + 5.3 \cdot 10^{-8} G^7_{-34}$ $\varphi_4 = 0.4 I$

Table 3. MP regulation and the calculated insulin dynamics related to four of the considered data-sets ($\tau = 2$ min).

5 Conclusions and ongoing work

The main goal of this work was to study the possible application of MP systems as an alternative to model the intravenous glucose tolerance test. In Section 2 we briefly described the test, while Section 3 reviewed two mathematical models which had the most important impacts in diabetology and analysed their limits and drawbacks. In Section 4 we proposed the use of Metabolic P systems to model

the IVGTT data-sets by combining some principles of MP systems with statistical techniques to obtain MP models of IVGTT. Our preliminary results and analysis suggest that glucose-insulin metabolism needs a careful evaluation which makes evident different aspects related to different subjects. MP models seem to provide comprehensive tools for discovering personalized glucose-insulin dynamics. Further analysis should permit to characterize the differentiation between subjects by considering physiological parameters such as the height, the weight, the work, the sport activity, and so on. Despite these differences, we are working in order to point out common features in the regulation governing the release of insulin. Our regression approach allows us a quantitative analysis which could highlight results which have been only theorized during the development of the differential models.

References

1. A. D. Aczel, and J. Sounderpandian. Complete Business Statistics. Mc Graw Hill, International Edition, 2006.
2. R.N. Bergman, D.T. Finegood, and M. Ader. Assessment of insulin sensitivity in vivo. *Endocr Rev*, 6(1):45–86, 1985.
3. R.N. Bergman, Y.Z. Ider, C.R. Bowden, and C. Cobelli. Quantitative estimation of insulin sensitivity. *Am J Physiol Endocrinol Metab*, 236(6):667–677, 1979.
4. A. Boutayeb and A. Chetouani. A critical review of mathematical models and data used in diabetology. *Biomedical engineering online*, 5:43+, 2006.
5. A. Castellini, G. Franco, and R. Pagliarini. Data analysis pipeline from laboratory to MP models. *Natural Computing*, 10(1):55–76, 2011.
6. E. Cerasi. Insulin deficiency and insulin resistance in the pathogenesis of niddm: is a divorce possible? *Diabetologia*, 38(8):992–997, 1995.
7. C. Cobelli and A. Mari. Validation of mathematical models of complex endocrine-metabolic systems. a case study on a model of glucose regulation. *Medical and Biological Engineering and Computing*, 21(4):390–399, 1983.
8. J.M. Cushing. *Integrodifferential equations and delay models in population dynamics*. Lecture notes in biomathematics. Springer-Verlag, 1977.
9. M. Derouich and A. Boutayeb. The effect of physical exercise on the dynamics of glucose and insulin. *Journal of Biomechanics*, 35(7):911 – 917, 2002.
10. A. De Gaetano and O. Arino. Mathematical modelling of the intravenous glucose tolerance test. *Journal of Mathematical Biology*, 40(2):136–168, 2000.
11. N. Draper and H. Smith. Applied Regression Analysis, 2nd Edition. John Wiley & Sons, New York, 1981.
12. R.R. Hocking. The Analysis and Selection of Variables in Linear Regression. *Biometrics* 32, 1976.
13. J. Li, Y. Kuang, and B. Li. Analysis of ivgtt Glucose-Insulin Interaction Models with time delay. *Discrete and Continuous Dynamical Systems Series B*, 1(1):103–124, 2001.
14. A. Makroglou, J. Li, and Y. Kuang. Mathematical models and software tools for the glucose-insulin regulatory system and diabetes: an overview. *Appl. Numer. Math.*, 56(3):559–573, 2006.
15. V. Manca. Metabolic P systems. *Scholarpedia* 5(3):9273, 2010.

16. V. Manca. Fundamentals of Metabolic P Systems. In [29], chapter 19, Oxford University Press, 2010.
17. V. Manca. Log-Gain Principles for Metabolic P Systems. In Condon, A. et al. (eds), Algorithmic Bioprocesses, Natural Computing Series, chapter 28, pp. 585–605, Springer-Verlag, 2009.
18. V. Manca, L. Bianco, and F. Fontana. Evolutions and Oscillations of P systems: Theoretical Considerations and Application to biological phenomena. In Membrane Computing, WMC 2004, LNCS 3365, pp. 63–84, Springer, 2005.
19. V. Manca and L. Marchetti. Log-Gain Stoichiometic Stepwise regression for MP systems. *International Journal of Foundations of Computer Science*, 22(1):97–106, 2011.
20. V. Manca and L. Marchetti. Goldbeters Mitotic Oscillator Entirely Modeled by MP Systems. Gheorghe M. et al. (Eds.): CMC 2010, LNCS 6501, pp. 273–284, Springer-Verlag Berlin Heidelberg, 2010.
21. V. Manca and L. Marchetti. Metabolic approximation of real periodical functions. *The Journal of Logic and Algebraic Programming* 79:363–373, 2010.
22. V. Manca, R. Pagliarini, and S. Zorzan. A photosynthetic process modelled by a metabolic P system. *Natural Computing*, 8(4):847–864, 2009.
23. A. Mari. Mathematical modeling in glucose metabolism and insulin secretion. *Curr Opin Clin Nutr Metab Care*, 5(5):495–501, 2002.
24. F. H. Martini. *Fundamentals of Anatomy and Physiology.* Benjamin Cummings, 8 edition, 2008.
25. A. Mukhopadhyay, A. De Gaetano, and O. Arino. Modelling the intravenous glucose tolerance test: A global study for single-distributed-delay model. *Discrete and Continuous Dynamical Systems Series B*, 4(2):407–417, 2004.
26. National Diabetes Data Group. Classification and diagnosis of diabetes mellitus and other categories of glucose intolerance. *Diabetes*, 28(28):1039–1057, 1979.
27. G. Pacini and R. N. Bergman. Minmod: a computer program to calculate insulin sensitivity and pancreatic responsivity from the frequently sampled intravenous glucose tolerance test. *Computer Methods and Programs in Biomedicine*, 23(2):113 – 122, 1986.
28. G. Păun. Membrane Computing. An Introduction. Springer, 2002.
29. G. Păun, G. Rozenberg, and A. Salomaa (eds): Oxford Handbook of Membrane Computing. Oxford University Press, 2010.
30. K. I. Rother. Diabetes treatment bridging the divide. *The New England Journal of Medicine*, 356(15):1499–1501.
31. G. Segre, G.L. Turco, and Vercellone G. Modeling blood glucose and insulin kinetics in normal, diabetic and obese subjects. *Diabetes*, 22(2):94–103, 1973.
32. G. Toffolo, R.N. Bergman, D.T. Finegood, C.R. Bowden, and C. Cobelli. Quantitative estimation of beta cell sensitivity to glucose in the intact organism: a minimal model of insulin kinetics in the dog. *Diabetes*, 29(12):979–990, 1980.
33. S. Wild, G. Roglic, A. Green, R. Sicree, and H. King. Global prevalence of diabetes. *Diabetes Care*, 27(5):1047–1053, 2004.

BFS Solution for Disjoint Paths in P Systems

Radu Nicolescu and Huiling Wu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
r.nicolescu@auckland.ac.nz, hwu065@aucklanduni.ac.nz

Summary. This paper continues the research on determining a maximum cardinality set of edge- and node-disjoint paths between a source cell and a target cell in P systems. We review the previous solution [3], based on depth-first search (DFS), and we propose a faster solution, based on breadth-first search (BFS), which leverages the parallel and distributed characteristics of P systems. The runtime complexity shows that, our BFS-based solution performs better than the DFS-based solution, in terms of P steps.

1 Introduction

P systems is a bio-inspired computational model, based on the way in which chemicals interact and cross cell membranes, introduced by Păun [16]. The essential specification of a P system includes a membrane structure, objects and rules. All cells evolve synchronously by applying rules in a non-deterministic and (potentially maximally) parallel manner. Thus, P systems is a strong candidate as a model for distributed and parallel computing.

Given a digraph G and two nodes, s and t, the disjoint paths problem aims to find the maximum number of s-to-t edge- or node-disjoint paths. There are many important applications that need to find alternative paths between two nodes, in all domains. Alternative paths are fundamental in biological remodelling, e.g., of nervous or vascular systems. Multipath routing can use all available bandwidth in computer networks. Disjoint paths are sought in streaming multi-core applications that are bandwidth sensitive to avoid sharing communication links between processors [17]. The maximum matching problem in a bipartite graph can also be transformed to the disjoint paths problem. In case of non-complete graphs, Byzantine Agreement requires at least $2k + 1$ node-disjoint paths, between each pair of nodes to ensure that a distributed consensus can occur, with up to k failures [9].

It is interesting to design a native P system solution for the disjoint path problem. In this case, the input graph is the P system structure itself, not as data to a program. Also, the system is fully distributed, i.e. there is no central node and only local channels (between structural neighbours) are allowed. In 2010, Dinneen,

Kim and Nicolescu [3] proposed the first P solution, as a distributed version of the Ford-Fulkerson algorithm, based on depth-first search (DFS). This solution searches by visiting nodes sequentially, which is not always efficient. To exploit the parallel potential of P systems, we propose a faster P system solution—a distributed version of the Edmonds-Karp algorithm, which concurrently searches as many paths as possible in breadth-first search (BFS).

This paper is organized as follows. Section 2 defines a simplified P system, general enough to cover most basic families. Section 3 describes the disjoint paths problem and the strategies for finding disjoint paths in digraphs. Section 4 discusses the specifics of the disjoint paths problem in P systems. Section 5 reviews the previous DFS-based solution [3] and sets out our faster BFS-based solution. Section 6 presents the P system rules for the disjoint paths algorithm using BFS. Section 7 compares the performance of the BFS-based and DFS-based algorithms, in terms of P steps, and the relative performance of the BFS-based solution simulation on sequential vs. parallel (multi-core) hardware. Finally, Section 8 summarizes our work and highlights future work.

2 Preliminary

Essentially, a static P system is specified by the membrane structure, objects and rules. The membrane structure can be modeled as: a rooted tree (cell-like P systems [16]), a directed acyclic graph (hyperdag P systems [11], [12], [13]), or in a more general case, an arbitrary digraph (neural P systems [10], [14]). Usually, the objects are symbols from a given alphabet, but one can also consider strings or other more complex structures (such as tuples). P systems combine rewriting rules that change objects in the region and communication rules that move objects across membranes. Here, we define a simple P system, with *priorities*, *promoters* and *duplex* channels as a system, $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_n, \delta)$, where:

1. O is a finite non-empty alphabet of *objects*;
2. $\sigma_1, \ldots, \sigma_n$ are cells, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$, $1 \leq i \leq n$, where:
   - Q_i is a finite set of *states*;
   - $s_{i,0} \in Q_i$ is the *initial state*;
   - $w_{i,0} \in O^*$ is the *initial multiset* of objects;
   - R_i is a finite *ordered* set of rewriting/communication *rules* of the form: $s\ x \rightarrow_\alpha s'\ x'\ (y)_{\beta|z}$, where: $s, s' \in Q_i$, $x, x', y, z \in O^*$, $\alpha \in \{min, max\}$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$.
3. δ is a set of *digraph* arcs on $\{1, 2, \ldots, n\}$, without symmetric arcs, representing *duplex* channels between cells.

The membrane structure is a digraph with duplex channels, so parents can send messages to children *and* children to parents, but the disjoint paths strictly follow the parent-child direction. Rules are prioritized and are applied in *weak priority* order [15].

The general form of a rule, which transforms state s to state s', is $s\ x \rightarrow_{\alpha} s'\ x'\ (y)_{\beta_\gamma}|_z$. This rule consumes multiset x, and then (after all applicable rules have consumed their left-hand objects) produces multiset x', in the same cell ("*here*"). Also, it produces multiset y and sends it, by *replication*, to all parents ("*up*"), to all children ("*down*"), or to all parents and children ("*up and down*"), according to the value of target indicator $\beta \in \{\uparrow, \downarrow, \updownarrow\}$ (effectively, here we use the *repl* communication mode, exclusively). $\alpha \in \{min, max\}$ describes the rewriting mode. In the *minimal* mode, an applicable rule is applied exactly once. In the *maximal* mode, an applicable rule is used as many times as possible and all rules with the same states s and s' can be applied in the maximally parallel manner. Finally, the optional z indicates a multiset of promoters, which are not consumed, but are required, when determining whether the rule can be applied.

3 Disjoint Paths

Given a *digraph*, $G = (V, E)$, a *source* node, $s \in V$, and a *target* node, $t \in V$, the edge- and node-disjoint paths problem looks for one of the largest sets of edge- and node-disjoint s-to-t paths. A set of paths is *edge-disjoint* or *node-disjoint* if they have no common arc or no common intermediate node. Note that node-disjoint paths are also edge-disjoint paths, but the converse is not true. Cormen et al. [1] give a more detailed presentation of the topics discussed in this section.

Figure 1 (a) shows two node-disjoint paths from 0 to 6, i.e. 0.3.6 and 0.1.4.6, which are also edge-disjoint. In this scenario, this is the maximum number of node-disjoint paths one can find. However, one could add to this set another path, 0.2.3.5.6, shown in Figure 1 (b), to obtain a set of three edge-disjoint paths.

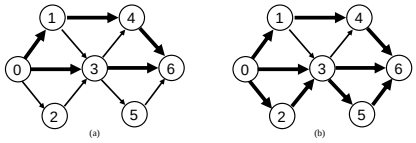


Fig. 1. Node- and edge- disjoint paths.

The maximum edge-disjoint paths problem can be transformed to a maximum flow problem by assigning unit capacity to each edge [5]. Given a set of already *established* edge- or node-disjoint paths P, we recall the definition of the *residual* digraph $G_r = (V_r, E_r)$:

- $V_r = \mathrm{V}$ and
- $E_r = (E \setminus E_P) \cup E'_P$, where E_P is the set of arcs (u, v) that appear in the P paths and $E'_P = \{(v, u) \mid (u, v) \in E_P\}$.

Briefly, the residual digraph is constructed by reversing the already established path arcs. An *augmenting* path is an s-to-t path in the residual digraph, G_r.

Augmenting paths are used to extend the existing set of established disjoint paths. If an augmenting arc reverses an existing path arc (also known as a *push-back* operation), then these two arcs "cancel" each other, due to zero total flow, and are discarded. The remaining path fragments are relinked to construct an extended set of disjoint paths. This round is repeated, starting with the new and larger set of established paths, until no more augmenting paths are found. A more detailed construction appears in Ford and Fulkerson maximal flow algorithm [5].

Example 1. Figure 2 illustrates a residual digraph and an augmenting path: (a) shows a digraph, where two edge-disjoint paths, 0.1.4.7 and 0.2.5.7, are present; (b) shows the residual digraph, formed by reversing path arcs; (c) shows an augmenting path, 0.3.5.2.6.7, which uses a reverse arc, $(5, 2)$; (d) discards the cancelling arcs, $(2, 5)$ and $(5, 2)$; (e) relinks the remaining path fragments, 0.1.4.7, 0.2, 5.7, 0.3.5 and 2.6.7, resulting in now three edge-disjoint paths, 0.1.4.7, 0.2.6.7 and 0.3.5.7.

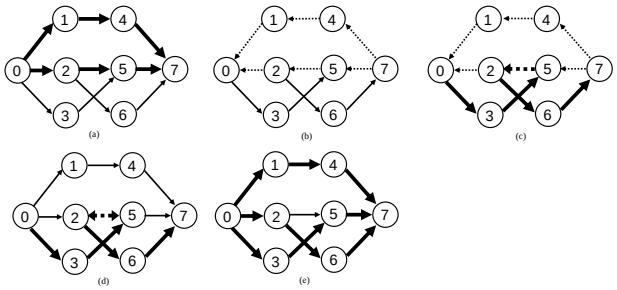


Fig. 2. Finding an augmenting path in the residual digraph.

The search for augmenting paths uses a search algorithm such as DFS (e.g., the Ford-Fulkerson algorithm) or BFS (e.g., the Edmonds-Karp algorithm). A *search path* in the residual graph (also known as a *tentative augmenting path*) starts from the source node and tries to reach the target node. A *successful search path* becomes a new *augmenting path* and is used (as previously explained) to increase the number of disjoint paths. Conceptually, this solves the edge-disjoint paths problem (at a high level). However, the node-disjoint paths require additional refinements—usually by *node splitting* [8]. Each *intermediate* node, v, is split into an *entry* node, v_1, and an *exit* node, v_2, linked by an arc (v_1, v_2). Arcs that in the original digraph, G, were directed into v are redirected into v_1 and arcs that were directed out of v are redirected out of v_2. Figure 3 illustrates this node-splitting procedure: (a) shows the original digraph and (b) the modified digraph, where all intermediate nodes are split—this is a bipartite digraph.

4 Disjoint Paths in P Systems

Classical algorithms use the digraph as data and keep global information. In contrast, our solutions are *fully distributed*. There is no central cell to convey global information among all cells, i.e. cells only communicate with their neighbors via local channels (between structural neighbours).

Unlike traditional programs, which keep full path information globally, our P systems solution records paths predecessors and successors locally in each cell, similar to distributed routing tables in computer networks. To construct such routing indicators, we assume that each cell σ_i is "blessed" with a unique *cell ID* object, ι_i, functioning as a *promoter*.

Although many versions of P systems accept cell division, we feel that this feature should not be used here and we intentionally discard it. Rather than actually splitting the intermediate P cells, we simulate this by ad-hoc cell rules. This approach could be in other distributed networks, where nodes cannot be split [3]. Essentially, node splitting prevents more than one unit flow to pass through an intermediate node [8].

In our case, node splitting can be simulated by: (i) constraining in and out flow capacities to one and (ii) having two *visited* markers for each cell, one for a *virtual entry* node and another for a *virtual exit* node, extending the visiting idea of classical search algorithms. Figure 3 illustrates a scenario when one cell, y, is visited *twice*, first on its entry and then on its exit node [3]. Assume that path $\pi = s.x.y.z.t$, is established. Consider a search path, τ, starting from cell, s, and reaching cell, y, in fact, y's entry node. This is allowed and y's entry node is marked as visited. However, to constrain its in-flow to one, y can only push-back τ on its in-flow arc, (x, y). Cell x's exit node becomes visited, x's out-flow becomes zero and τ continues on x's outgoing arc, (x, z). When τ reaches cell z, z's entry node becomes visited and z pushes τ back on its in-flow arc, (y, z). Cell y's exit node becomes visited, y's out-flow becomes zero and τ continues on y's outgoing arc, (y, t). When no other outgoing arc is present, the cell needs to push-back from its exit node to its entry node, which is only possible if its entry node is not visited. Finally, the search path, τ, reaches the target, t, and becomes $\tau = s.y.x.z.t$. After removing cancelling arcs and relinking the remaining ones, we have two node-disjoint paths, $s.x.z.t$ and $s.y.t$.

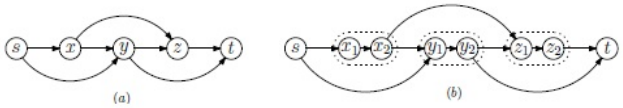


Fig. 3. Simulating node splitting [3].

5 Distributed DFS-based and BFS-based Solutions

As mentioned in Section 3, augmenting paths can be searched using DFS or BFS. Conceptually, DFS explores as far as possible along a single branch, before backtracking, while BFS explores as many branches as possible concurrently—P systems can exploit this parallelism.

5.1 Distributed DFS-based Strategy

Dinneen et al's DFS-based algorithms find disjoint paths in successive rounds [3].

Each round starts with a set of already *established disjoint paths*, which is empty at the start of the first round. The *source* cell, σ_s, starts to explore one of the untried branches. If the search path reaches the *target* cell, σ_t, it *confirms* to σ_s that a new augmenting paths was found; otherwise, it *backtracks*. While moving towards σ_s, the confirmation reshapes the existing paths and the newly found augmenting path, i.e. discarding cancelling arcs and relinking the rest, building a larger set of paths,

If σ_s receives the confirmation (one search path was successful, i.e. a new augmenting path was found), it broadcasts a *reset signal*, to prepare the next round. Otherwise, if the search fails, σ_s receives the backtrack. If there is an untried branch, the round is repeated. Otherwise, σ_s broadcasts a *finalize signal* to all cells and the search terminates.

This search algorithm is similar to a classical distributed DFS. Other more efficient distributed DFS algorithms [18] can be considered, but we do not follow this issue here.

5.2 Distributed BFS-based Strategy

Our BFS-based algorithms also work in successive rounds:

Each round starts with a set of already established disjoint paths, which is empty at the start of the first round. The source cell, σ_s, broadcasts a "wave", to find new augmenting paths. Current "frontier"cells send out *connect signals*. The cells which receive and *accept* these connect signals become the new frontier, by appending themselves at the end of current search paths. The advancing wave periodically sends *progress indicators* back to the source: (a) *connect acknowledgments* (at least one search path is still extending) and (b) *path confirmations* (at least one search path was successful, i.e. at least a new augmenting path was found). While travelling towards the source, each path confirmation reshapes the existing paths and the newly found augmenting path, creating a larger set of paths.

If no progress indicator arrives in the expected time, σ_s assumes that the search round ends. If at least one search path was successful (at least one augmenting path was found), σ_s broadcasts a *reset signal*, which prepares the next round, by resetting all cells (except the target). Otherwise, σ_s broadcasts a *finalize signal* to all cells and the search terminates.

In each round, an *intermediate* cell, σ_i, can be visited only once. Several search paths may try to visit the same intermediate cell *simultaneously*, but only one of them succeeds. Figure 4 (a) shows such a scenario: cells 1, 2 and 3 try to connect cell 4, in the same step; but only cell 1 succeeds, via arc $(1, 4)$. This *choice* operation is further described in Section 6.

The target cell, σ_t, faces a subtle decision problem. When several search paths arrive, simultaneously or sequentially, σ_t must quickly decide which augmenting path can be established and which one must be ignored (in the current round). We solve this problem using a *branch-cut* strategy. Given a search path, τ, its *branch ID* is the cell ID of its first intermediate cell after the source, taken by τ. Figure 4 (b) shows four potential paths arriving at cell 6: $\pi = 0.1.6$, $\tau_1 = 0.1.3.6$, $\tau_2 = 0.1.5.6$ and $\tau_3 = 0.2.4.6$; their branch IDs are 1, 1, 1 and 2, respectively. Paths π, τ_1 and τ_2 share the same branch ID, 1, and are incompatible. The following result is straightforward:

Proposition 1. *In any search round, search paths which share the same branch ID are incompatible; only one of them can be accepted.*

Therefore, the target cell accept or reject decision is based on branch ID. These *branch ID* operations are further described in Section 6.

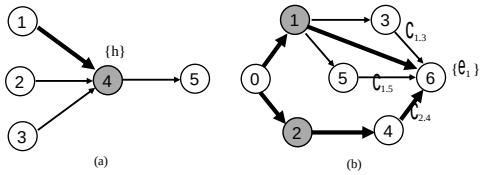


Fig. 4. BFS challenges. (a) A choice must be made between several search paths connecting the same cell (4), (b) Search paths sharing the same branch ID are incompatible.

6 P System Rules for Disjoint Paths Using BFS

The P system rules for edge- and node-disjoint paths are slightly different, due to the simulated node-splitting approach, but the basic principle is the same. We first discuss the edge-disjoint and then the changes required to cover the node-disjoint.

6.1 Rules for Edge-disjoint Paths

Algorithm 1 (P system algorithm for edge-disjoint paths)

Input: All cells start with the *same set of rules* and *without any topological awareness* (they do not even know their local neighbours). All cells start in the

same initial state. Initially, each cell, σ_i, contains a *cell ID* object, ι_i, which is *immutable* and used as a *promoter*. Additionally, the source cell, σ_s, and the target cell, σ_t, are decorated with objects, a and z, respectively.

Output: All cells end in the *same final state*. On completion, all cells are *empty*, with the following exceptions: (1) The source cell, σ_s, and the target cell, σ_t, are still decorated with objects, a and z, respectively; (2) The cells on *edge-disjoint paths* contain path link objects, for *predecessors*, p_j, and for *successors*, s_k.

We use the following six states: S_0, the initial state; S_1, the quiescent state; S_2, the frontier state; S_3, for previous frontier cells; S_4, the final state; and S_5, a special state for the target cell.

Initially, all cells are in the initial state, S_0. When each cell produces a catalyst-like object, it enters the quiescent state, S_1. When cells in S_1 accept connect signals, they enter the frontier state, S_2, except the target which changes directly to S_5. Cells on the frontier send connect signals to neighbors and then change to S_3, to receive and relay progress indicators. Specifically, the target remains in S_5, after accepting the first connect signal (because it is always waiting to be connected), until it receives the finalize signal. When the search finishes, all cells transit to the final state, S_4. Figure 5 shows all state transitions.

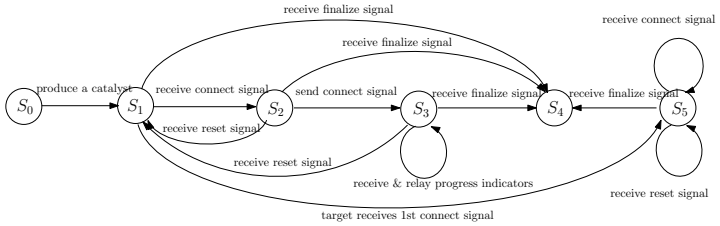


Fig. 5. State-chart of BFS-based algorithm.

We use these symbols to describe our edge-disjoint implementation:

- a indicates the source cell.
- z indicates the target cell.
- d indicates, in the source cell, that an augmenting path was found in the current round (it appears in the source cell).
- e_j records, in the target cell, the branch ID of a successful augmenting path (i.e. σ_j is the first cell after the source, in this augmenting path).
- c_s is the connect signal sent by the source cell, σ_s, to its children.
- $c_{j.k}$ is the connect signal sent by an intermediate cell, σ_k, to its children; j is the branch ID.
- $l_{j.k}$ is the connect signal sent by an intermediate cell, σ_k, to its parents; j is the branch ID.
- r_j is the connect acknowledgment sent to cell, σ_j.

- $f_{j.k}$ is the path confirmation of a successful augmenting path, sent by cell σ_j to cell σ_k.
- h is a catalyst object in each cell.
- o is a signal broadcast by the source cell, σ_s, to make each cell produce one catalyst object.
- u indicates the first intermediate cell after the source, which is produced on receiving the connect signal, c_s.
- b is the reset signal which starts a new round.
- g is the finalize signal which terminates the search.
- t_j indicates that cell σ_j is a predecessor on a search path (recorded when a cell accepts a connect signal).
- p_j is a disjoint path predecessor (recorded when a successful augmenting path is confirmed).
- s_j is a disjoint path successor (recorded when a successful augmenting path is confirmed).
- w, v implement a source cell timer to wait for the first response or confirmation.
- x, y implement another source cell timer to wait for the periodically relayed response or confirmation.

We next present the rules and briefly explain them.

0. Rules in state S_1:
   1 $S_0\ a \to_{\texttt{min}} S_1\ ah(o)_{\downarrow}$
   2 $S_0\ o \to_{\texttt{min}} S_1\ h(o)_{\downarrow}$
   3 $S_0\ o \to_{\texttt{max}} S_1$

1. Rules in state S_1:
   1 $S_1\ o \to_{\texttt{max}} S_1$
   2 $S_1\ d \to_{\texttt{max}} S_1$
   3 $S_1\ b \to_{\texttt{max}} S_1$
   4 $S_1\ e_j \to_{\texttt{max}} S_1$
   5 $S_1\ g \to_{\texttt{min}} S_4\ (g)_{\downarrow}$
   6 $S_1\ v \to_{\texttt{max}} S_1$
   7 $S_1\ w \to_{\texttt{max}} S_1$
   8 $S_1\ x \to_{\texttt{max}} S_1$
   9 $S_1\ y \to_{\texttt{max}} S_1$
   10 $S_1\ f_{j.k} \to_{\texttt{max}} S_1$
   11 $S_1\ t_j \to_{\texttt{max}} S_1$
   12 $S_1\ r_j \to_{\texttt{max}} S_1$
   13 $S_1\ a \to_{\texttt{min}} S_2\ a$
   14 $S_1\ c_j p_j \to_{\texttt{min}} S_1\ up_j$
   15 $S_1\ c_{j.k} p_k \to_{\texttt{min}} S_1\ p_k$
   16 $S_1\ zhc_{j.k} \to_{\texttt{min}} S_5\ zhp_k e_j(f_{i.k})_{\updownarrow}|_{\iota_i}$
   17 $S_1\ zhc_j \to_{\texttt{min}} S_5\ zhup_j(f_{i.j})_{\updownarrow}|_{\iota_i}$
   18 $S_1\ hl_{j.k} s_k \to_{\texttt{min}} S_2\ ht_k e_j s_k\ (r_k)_{\updownarrow}$

19 $S_1\ hc_j \rightarrow_{\texttt{min}} S_2\ hut_j\ (r_j)_{\updownarrow}$
20 $S_1\ hc_{j.k} \rightarrow_{\texttt{min}} S_2\ ht_k e_j\ (r_k)_{\updownarrow}$

2. Rules in state S_2:
1 $S_2\ b \rightarrow_{\texttt{min}} S_1(b)_{\downarrow}$
2 $S_2\ g \rightarrow_{\texttt{min}} S_4(g)_{\downarrow}$
3 $S_2\ ah \rightarrow_{\texttt{min}} S_3\ ahw(c_i)_{\downarrow}|_{\iota_i}$
4 $S_2\ he_j \rightarrow_{\texttt{min}} S_3\ he_j(l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
5 $S_2\ hu \rightarrow_{\texttt{min}} S_3\ hu(l_{i.i})_{\uparrow}\ (c_{i.i})_{\downarrow}|_{\iota_i}$
6 $S_2\ f_{j.k} \rightarrow_{\texttt{max}} S_2$
7 $S_2\ c_{j.k} \rightarrow_{\texttt{max}} S_2$
8 $S_2\ l_{j.k} \rightarrow_{\texttt{max}} S_2$

3. Rules for state S_3:
1 $S_3\ b \rightarrow_{\texttt{min}} S_1(b)_{\downarrow}$
2 $S_3\ g \rightarrow_{\texttt{min}} S_4(g)_{\downarrow}$
3 $S_3\ axyyf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
4 $S_3\ axyyr_i \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
5 $S_3\ axyyyf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
6 $S_3\ axyyyr_i \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
7 $S_3\ adxyyy \rightarrow_{\texttt{min}} S_1\ a(b)_{\downarrow}$
8 $S_3\ axyyy \rightarrow_{\texttt{min}} S_4\ a(g)_{\downarrow}$
9 $S_3\ awvv \rightarrow_{\texttt{min}} S_4\ a(g)_{\downarrow}$
10 $S_3\ awvf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
11 $S_3\ awvr_i \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
12 $S_3\ x \rightarrow_{\texttt{min}} S_3\ y$
13 $S_3\ t_j f_{k.i} \rightarrow_{\texttt{min}} S_3\ p_j s_k(f_{i.j})_{\updownarrow}|_{\iota_i}$
14 $S_3\ af_{j.i} \rightarrow_{\texttt{min}} S_3\ as_j|_{\iota_i}$
15 $S_3\ p_j s_j \rightarrow_{\texttt{min}} S_3$
16 $S_3\ r_i t_j \rightarrow_{\texttt{min}} S_3\ t_j(r_j)_{\updownarrow}|_{\iota_i}$
17 $S_3\ w \rightarrow_{\texttt{min}} S_3\ wv$
18 $S_3\ r_j \rightarrow_{\texttt{max}} S_3$
19 $S_3\ c_{j.k} \rightarrow_{\texttt{max}} S_3$
20 $S_3\ f_{j.k} \rightarrow_{\texttt{max}} S_3$
21 $S_3\ l_{j.k} \rightarrow_{\texttt{max}} S_3$

4. Rules for state S_4:
1 $S_4\ g \rightarrow_{\texttt{max}} S_4$
2 $S_4\ e_j \rightarrow_{\texttt{max}} S_4$
3 $S_4\ f_{j.k} \rightarrow_{\texttt{max}} S_4$
4 $S_4\ c_{j.k} \rightarrow_{\texttt{max}} S_4$
5 $S_4\ l_{j.k} \rightarrow_{\texttt{max}} S_4$
6 $S_4\ t_j \rightarrow_{\texttt{max}} S_4$
7 $S_4\ r_j \rightarrow_{\texttt{max}} S_4$
8 $S_4\ w \rightarrow_{\texttt{max}} S_4$

9 $S_4\ v \rightarrow_{\texttt{max}} S_4$
10 $S_4\ u \rightarrow_{\texttt{max}} S_4$
11 $S_4\ h \rightarrow_{\texttt{max}} S_4$
12 $S_4\ o \rightarrow_{\texttt{max}} S_4$

5. Rules for state S_5:
1 $S_5\ c_j p_j \rightarrow_{\texttt{min}} S_5\ p_j$
2 $S_5\ c_{j.k} \rightarrow_{\texttt{min}} S_5|_{e_j}$
3 $S_5\ c_{j.k} p_k \rightarrow_{\texttt{min}} S_5\ p_k$
4 $S_5\ hc_{j.k} \rightarrow_{\texttt{min}} S_5\ hp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
5 $S_5\ hc_j \rightarrow_{\texttt{min}} S_5\ hp_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
6 $S_5\ g \rightarrow_{\texttt{max}} S_4$
7 $S_5\ b \rightarrow_{\texttt{max}} S_5$
8 $S_5\ f_{j.k} \rightarrow_{\texttt{max}} S_5$
9 $S_5\ l_{j.k} \rightarrow_{\texttt{max}} S_5$
10 $S_5\ t_j \rightarrow_{\texttt{max}} S_5$
11 $S_5\ r_j \rightarrow_{\texttt{max}} S_5$
12 $S_5\ u \rightarrow_{\texttt{max}} S_5$

The following paragraphs outline how these rules are used by each major cell group: the source cell, frontier cells, other intermediate cells and the target cell.

Scripts for the source cell: In the initial state S_0, the source cell, σ_s, indicated by the special object a, starts by broadcasting an object, o, to all cells and enters S_1 (rule 0.1); each receiving cell creates a local catalyst-like object, h, and enters S_1 (rule 0.2).

Next, cell σ_s enters S_2 (rule 1.13) and starts the search wave via connection requests, c_s (rule 2.3). Then, the source cell σ_s changes to state S_3 and uses timers to wait (a) one step for the the *first* progress indicators (rules 3.10, 3.11, 3.17), and (b) two steps for further *relayed* progress indicators (rules 3.3, 3.4, 3.12). If no progress indicator arrives when the timer overflows, cell σ_s waits *one more step* (rules 3.5, 3.6). If still no expected progress indicator arrives, cell σ_s assumes the round has ended. If an augmenting path was found in the current round, σ_s broadcasts a reset signal b to reset all cells (except the target σ_t) to S_1 (rule 3.7). Otherwise, σ_s broadcasts a finalize signal, g, which prompts all cells to enter S_4 (rules 3.8, 3.9).

It is interesting to note why the source cell needs to wait for one more step, even when the timer overflows. An intermediate cell filters connect signals, using rules 1.14–15, which have higher priority than the rules to accept a connect signal, i.e. rules 1.18–20. The rules to accept a connect signal cannot apply in the same step because of the different target states. For example, in Figure 6, path 0.2.4.6.7.9 is found in the first round. In the second round, search paths 0.1.4 and 0.3.5 attempt to connect to cell 6. Cell 6 discards cell 4's connect signal, following the higher-priority rule 1.15 and then, in the next step, accepts cell 5's connect signal, using rule 1.20. In this case, the source cell needs an extra one-step delay, to receive the

relayed connect acknowledgment from cell 6. All unacceptable signals are discarded in one step, so a one-step delay is enough.

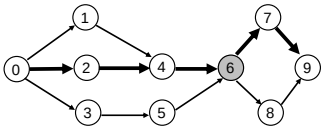


Fig. 6. A particular case requiring a delayed connect acknowledgment.

Scripts for a frontier cell: An intermediate cell, σ_i, if it is unvisited in this round, accepts exactly one connect signal and discards the rest; otherwise, it discards all connect signals. By accepting one connect signal, σ_i enters S_2 and becomes a frontier cell to send connect signals. When σ_i sends its connect signals, the frontier advances.

An intermediate cell, σ_i, may receive connect signals: (a) c_s, connect signals sent by the source cell, σ_s, to its children; (b) $c_{j.k}$, connect signals sent by a frontier cell, σ_k, to its children; (c) $l_{j.k}$, connect signals sent by a frontier cell, σ_k, to its parents. Received connect signals are checked for *acceptability*: (a) a c_s or $c_{j.k}$ connect signal is *acceptable* if it does not come from an established path predecessor, which corresponds to a forward operation (rules 1.14, 1.15, 1.19, 1.20); (b) a $l_{j.k}$ connect signal is *acceptable* if it comes from an established path successor, which corresponds to a push-back operation (rule 1.18).

Cell σ_i becomes a frontier cell by accepting either: (1) a connect signal, c_s, from σ_s (rules 1.14, 1.19), in this case, cell σ_i (a) generates an u, indicating that it is the first intermediate cell on the current search path (the first after cell σ_s); (b) records its predecessor on the search path, σ_s, as t_s; and (c) sends a connect acknowledgment, r_s, back to cell σ_s; or (2) a connect signal, $c_{j.k}$ or $l_{j.k}$ from σ_k (rules 1.15, 1.18, 1.20), in this case, cell σ_i (a) records the branch ID, j, as e_j; (b) records its predecessor on the search path, σ_k, as t_k; and (c) sends a connect acknowledgment, r_k, back to cell σ_k.

Then, as a frontier cell, σ_i sends connect signals to neighbors and changes to state S_3: (1) if cell σ_i is marked by an u object, it uses its own ID, i, as the branch ID to further generate connect signals, $c_{i.i}$ or $l_{i.i}$ (rule 2.5); (2) otherwise, σ_i uses the recorded e_j as the branch ID to further generate connect signals, $c_{j.i}$ or $l_{j.i}$ (rule 2.4).

Consider the scenario when several connect signals arrive *simultaneously* in an *unvisited* cell, σ_i (see Figure 4 (a)). Cell σ_i makes a (conceptually random) *choice* and selects exactly one of the acceptable connect signals, thus deciding which search path can follow through. To solve this choice problem, we use an object, h, which functions like a catalyst [15]. Object h is immediately consumed by the rule which *accepts* the connect signal, therefore no other connect signal is accepted (rules 1.16–20). Next, the catalyst, h, is recreated, but the cell also

changes its state, thus it cannot accept another connect signal (not in the same search round).

Scripts for other intermediate cell: A previous frontier cell, σ_i, relays progress indicators: connect acknowledgments, r_i (rule 3.16) and path confirmations, $f_{k.i}$ (rule 3.13). On receiving path confirmations, σ_i transforms a temporary path predecessor, t_j, into an established path predecessor, p_j, and records the path successors, s_k. In the next step, cell σ_i discards matching predecessor and successor objects (i.e. referring to the same cell), e.g., σ_i may already contain (from a previous round) another predecessor-successor pair, $p_{j'}$, $s_{k'}$. If $j = k'$, then p_j and $s_{k'}$ are deleted, as one end of the cancelling arc pair, (j, i) and (i, j); similarly, if $k = j'$, then s_k and $p_{j'}$ are deleted (rule 3.15).

Scripts for the target cell: The target cell, σ_t, accepts either (1) a connect signal from σ_s, c_s, if it does not come from an established path predecessor (rules 1.17, 5.1, 5.5), or (2) a connect signal from a frontier cell σ_k, $c_{j.k}$ (rules 1.16, 5.2, 5.3, 5.4), which indicates the different branch ID (rule 5.2) and does not come from an established path predecessor (rule 5.3). In case (1), cell σ_t: (a) generates an u, indicating that it is the second cell on a search path (the first after cell σ_s); (b) records its predecessor on the search path, σ_s, as p_s; and (c) sends a path confirmation $f_{t.s}$, back to cell σ_s. In case (2), cell σ_t: (a) records the branch ID, j, as e_j; (b) records its predecessor on the search path, σ_k, as p_k; and (c) sends a path confirmation, $f_{t.k}$, back to cell σ_k.

This branch-cut strategy is illustrated in Figure 4 (b). It shows an established path, $\pi = 0.1.6$, whose branch ID is recorded as e_1. Consider the fate of other search paths, $\tau_1 = 0.1.3.6$, $\tau_2 = 0.1.5.6$, and $\tau_3 = 0.2.4.6$, which attempt to reach the target 6, later in the same round. τ_1 sends the connect signal $c_{1.3}$, which is rejected. τ_2 sends the connect signal $c_{1.5}$, which is also rejected. τ_3 sends the connect signal $c_{2.4}$, which is accepted. To summarize, in this example round, two augmenting paths are established, π and τ_3; other attempts are properly ignored.

It is important that recording objects e_i are used as *promoters*, which enable rules, without being consumed [7]. Otherwise, objects e_i can be consumed before completing their role; e.g., the rejection of τ_1 would consume e_1 and there would be nothing left to reject τ_2.

Example 2. Table 7 shows Algorithm 1 tracing fragments for stages (a), (c) and (e) of Figure 2, illustrating how our P system solution works. In all stages, each cell, σ_i, contains a promoter object, ι_i, as the cell ID; the source cell, σ_0, and the target cell, σ_7, are decorated by objects, a and z, respectively. The catalyst object, h, remains in each cell after it is produced, until the cell enters the final state, S_4.

In stage 1(a), the two established paths, 0.1.4.7 and 0.2.5.7, are recorded by the following cell contents: $\sigma_0 : \{s_1, s_2\}$, $\sigma_1 : \{p_0, s_4\}$, $\sigma_2 : \{p_0, s_5\}$, $\sigma_4 : \{p_1, s_7\}$, $\sigma_5 : \{p_2, s_7\}$, $\sigma_7 : \{p_4, p_5\}$. In the source cell σ_0, xy^3 is a timer to wait for the relayed progress indicators, which currently overflows. The object d indicates that an augmenting path was found in the current round, so in the next step, the source cell, σ_0, broadcasts a reset signal to all cells to start a new round. Cells σ_1, σ_2, and

Table 7. Algorithm 1 tracing fragments for stages (a), (c) and (e) of Figure 2.

Stage\Cell	σ_0	σ_1	σ_2	σ_3
1(a)	$S_3\ \iota_0 adhs_1s_2xy^3$	$S_3\ \iota_1 hp_0s_4u$	$S_3\ \iota_2 hp_0s_5u$	$S_3\ \iota_3 ht_0u$
1(c)	$S_3\ \iota_0 ahs_1s_2xy$	$S_1\ \iota_1 hp_0s_4u^2$	$S_3\ \iota_2 e_3 hp_0r_3s_5t_5u^2$	$S_3\ \iota_3 hr_3t_0u^2$
1(e)	$S_4\ \iota_0 as_1s_2s_3$	$S_4\ \iota_1 p_0s_4$	$S_4\ \iota_2 p_0s_6$	$S_4\ \iota_3 p_0s_5$
Stage\Cell	σ_4	σ_5	σ_6	σ_7
1(a)	$S_3\ \iota_4 e_1hp_1s_7$	$S_3\ \iota_5 e_2hp_2s_7$	$S_3\ \iota_6 e_2ht_2$	$S_5\ \iota_7 e_1e_2hp_4p_5z$
1(c)	$S_1\ \iota_4 f_{7.6}hp_1s_7$	$S_3\ \iota_5 e_3f_{7.6}hp_2s_7t_3$	$S_3\ \iota_6 e_3f_{7.6}ht_2$	$S_5\ \iota_7 e_1e_2e_3hp_4p_5p_6r_3z$
1(e)	$S_4\ \iota_4 p_1s_7$	$S_4\ \iota_5 p_3s_7$	$S_4\ \iota_6 p_2s_7$	$S_4\ \iota_7 p_4p_5p_6z$

σ_3 have objects, u, indicating that they are the first intermediate cells after the source, while cells $\sigma_4, \sigma_5, \sigma_6$ contain objects, e_j, which mean they should include j as the branch ID when sending connect signals. The target cell, σ_7, records the already used branch IDs, e_1 and e_2.

In stage 1(c), the successful search path 0.3.5.2.6.7 is recorded as: $\sigma_3 : \{t_0\}$, $\sigma_5 : \{t_3\}$, $\sigma_2 : \{t_5\}$, $\sigma_6 : \{t_2\}$, $\sigma_7 : \{p_6\}$ (the target records p_6 directly). The target cell σ_7 also records the branch ID of the newly successful path, e_3, and sends back a path confirmation $f_{7.6}$ to all its neighbors. In cell σ_3, the objects, r_3 and t_0, indicate that the connect acknowledgment needs to be relayed to the source cell σ_0. Thus, in the next step, cell σ_0 receives a connect acknowledgment from cell σ_3 and resets the timer.

In stage 1(e), all cells enter the final state S_4 and there are three established paths, 0.1.4.7, 0.2.6.7 and 0.3.5.7, which are recorded as: $\sigma_0 : \{s_1, s_2, s_3\}$, $\sigma_1 : \{p_0, s_4\}$, $\sigma_2 : \{p_0, s_6\}$, $\sigma_3 : \{p_0, s_5\}$, $\sigma_4 : \{p_1, s_7\}$, $\sigma_5 : \{p_3, s_7\}$, $\sigma_6 : \{p_2, s_7\}$, $\sigma_7 : \{p_4, p_5, p_6\}$.

The preceding arguments indicate a bisimulation relation between our BFS-based algorithm and the classical Edmonds and Karp BFS-based algorithm for edge-disjoint paths [4]. The following theorem encapsulates all these arguments:

Theorem 1. *When Algorithm 1 terminates, path predecessor and successor objects listed in its output section indicate a maximal cardinality set of edge-disjoint paths.*

6.2 Rules for Node-disjoint Paths

Algorithm 2 (P system algorithm for node-disjoint paths)

Input: As in the edge-disjoint paths algorithm of Algorithm 1.

Output: Similar to in the edge-disjoint paths algorithm. However, the predecessor and successor objects indicate *node-disjoint paths*, instead of edge-disjoint paths.

To simulate node splitting, the node-disjoint version uses additional symbols (as before, rules assume that cell σ_i is the current cell):

- m indicates that the "entry node is visited".
- n indicates that the "exit node is visited".

- q indicates that this cell's in-flow and out-flow is one (or, equivalently, that this cell is in an already established or confirmed path).
- $t_{j.k}$ indicates cell σ_i's predecessor, σ_j, on a search path, recorded after it receives the connect acknowledgment from cell σ_i's successor, σ_k (before receiving this acknowledgment, σ_i's predecessor is temporarily recorded as t_j.)
- $r_{j.k}$ is a connect acknowledgment sent by cell σ_j to cell σ_k.

0. Rules in state S_1:
   1 $S_0\ a \rightarrow_{\texttt{min}} S_1\ ah(o)_{\downarrow}$
   2 $S_0\ o \rightarrow_{\texttt{min}} S_1\ h(o)_{\downarrow}$
   3 $S_0\ o \rightarrow_{\texttt{max}} S_1$

1. Rules in state S_1:
   1 $S_1\ o \rightarrow_{\texttt{max}} S_1$
   2 $S_1\ d \rightarrow_{\texttt{max}} S_1$
   3 $S_1\ b \rightarrow_{\texttt{max}} S_1$
   4 $S_1\ e_j \rightarrow_{\texttt{max}} S_1$
   5 $S_1\ g \rightarrow_{\texttt{min}} S_4\ (g)_{\downarrow}$
   6 $S_1\ v \rightarrow_{\texttt{max}} S_1$
   7 $S_1\ w \rightarrow_{\texttt{max}} S_1$
   8 $S_1\ u \rightarrow_{\texttt{max}} S_1$
   9 $S_1\ m \rightarrow_{\texttt{max}} S_1$
   10 $S_1\ n \rightarrow_{\texttt{max}} S_1$
   11 $S_1\ f_{j.k} \rightarrow_{\texttt{max}} S_1$
   12 $S_1\ t_{j.k} \rightarrow_{\texttt{max}} S_1$
   13 $S_1\ t_j \rightarrow_{\texttt{max}} S_1$
   14 $S_1\ r_{j.k} \rightarrow_{\texttt{max}} S_1$
   15 $S_1\ a \rightarrow_{\texttt{min}} S_2\ a$
   16 $S_1\ c_j p_j \rightarrow_{\texttt{min}} S_1\ p_j$
   17 $S_1\ c_{j.k} p_k \rightarrow_{\texttt{min}} S_1\ p_k$
   18 $S_1\ zhc_{j.k} \rightarrow_{\texttt{min}} S_5\ zhp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
   19 $S_1\ zhc_j \rightarrow_{\texttt{min}} S_5\ zhp_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
   20 $S_1\ hl_{j.k} s_k \rightarrow_{\texttt{min}} S_2\ ht_k e_j s_k n\ (r_{i.k})_{\updownarrow}|_{\iota_i}$
   21 $S_1\ hc_{j.k} q \rightarrow_{\texttt{min}} S_2\ ht_k e_j mq\ (r_{i.k})_{\updownarrow}|_{\iota_i}$
   22 $S_1\ hc_j \rightarrow_{\texttt{min}} S_2\ hut_j\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
   23 $S_1\ hc_{j.k} \rightarrow_{\texttt{min}} S_2\ ht_k e_j\ (r_{i.k})_{\updownarrow}|_{\iota_i}$

2. Rules in state S_2:
   1 $S_2\ b \rightarrow_{\texttt{min}} S_1 (b)_{\downarrow}$
   2 $S_2\ g \rightarrow_{\texttt{min}} S_4 (g)_{\downarrow}$
   3 $S_2\ ah \rightarrow_{\texttt{min}} S_3\ ahw(c_i)_{\downarrow}|_{\iota_i}$
   4 $S_2\ he_j m \rightarrow_{\texttt{min}} S_3\ he_j m\ (l_{j.i})_{\uparrow}|_{\iota_i}$
   5 $S_2\ he_j n \rightarrow_{\texttt{min}} S_3\ he_j n\ (l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
   6 $S_2\ he_j \rightarrow_{\texttt{min}} S_3\ he_j\ (l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
   7 $S_2\ hu \rightarrow_{\texttt{min}} S_3\ hu(l_{i.i})_{\uparrow}\ (c_{i.i})_{\downarrow}|_{\iota_i}$

8 $S_2\ f_{j.k} \rightarrow_{\texttt{max}} S_2$
9 $S_2\ c_{j.k} \rightarrow_{\texttt{max}} S_2$
10 $S_2\ l_{j.k} \rightarrow_{\texttt{max}} S_2$

3. Rules in state S_3:
1 $S_3\ b \rightarrow_{\texttt{min}} S_1 (b)_{\downarrow}$
2 $S_3\ g \rightarrow_{\texttt{min}} S_4 (g)_{\downarrow}$
3 $S_3\ hml_{j.k} s_k \rightarrow_{\texttt{min}} S_3\ hmnt_k e_j s_k\ (r_{i.k})_{\updownarrow}|_{\iota_i}$
4 $S_3\ he_j mn \rightarrow_{\texttt{min}} S_3\ hwe_j\ (l_{j.i})_{\uparrow}\ (c_{j.i})_{\downarrow}|_{\iota_i}$
5 $S_3\ axyyf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
6 $S_3\ axyyr_{j.i} \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
7 $S_3\ axyyyf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
8 $S_3\ axyyyr_{j.i} \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
9 $S_3\ adxyyy \rightarrow_{\texttt{min}} S_1\ a\ (b)_{\downarrow}|_{\iota_i}$
10 $S_3\ axyyy \rightarrow_{\texttt{min}} S_4\ a\ (g)_{\downarrow}|_{\iota_i}$
11 $S_3\ awvv \rightarrow_{\texttt{min}} S_4\ a(g)_{\downarrow}|_{\iota_i}$
12 $S_3\ awvf_{j.i} \rightarrow_{\texttt{min}} S_3\ ads_j x|_{\iota_i}$
13 $S_3\ awvr_{j.i} \rightarrow_{\texttt{min}} S_3\ ax|_{\iota_i}$
14 $S_3\ x \rightarrow_{\texttt{min}} S_3\ xy$
15 $S_3\ t_{j.k} f_{k.i} \rightarrow_{\texttt{min}} S_3\ p_j s_k q\ (f_{i.j})_{\updownarrow}|_{\iota_i}$
16 $S_3\ t_j f_{k.i} \rightarrow_{\texttt{min}} S_3\ p_j s_k q\ (f_{i.j})_{\updownarrow}|_{\iota_i}$
17 $S_3\ af_{j.i} \rightarrow_{\texttt{min}} S_3\ as_j|_{\iota_i}$
18 $S_3\ p_j s_j q \rightarrow_{\texttt{min}} S_3$
19 $S_3\ r_{k.i} t_{j.k} \rightarrow_{\texttt{min}} S_3\ t_{j.k}\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
20 $S_3\ t_j r_{k.i} \rightarrow_{\texttt{min}} S_3\ t_{j.k}\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
21 $S_3\ t_{j.l} r_{k.i} \rightarrow_{\texttt{min}} S_3\ t_{j.l} t_{j.k}\ (r_{i.j})_{\updownarrow}|_{\iota_i}$
22 $S_3\ w \rightarrow_{\texttt{min}} S_3\ wv$
23 $S_3\ ar_{j.i} \rightarrow_{\texttt{max}} S_3\ a|_{\iota_i}$
24 $S_3\ c_{j.k} \rightarrow_{\texttt{max}} S_3$
25 $S_3\ f_{j.k} \rightarrow_{\texttt{max}} S_3$
26 $S_3\ l_{j.k} \rightarrow_{\texttt{max}} S_3$

4. Rules in state S_4:
1 $S_4\ g \rightarrow_{\texttt{max}} S_4$
2 $S_4\ e_j \rightarrow_{\texttt{max}} S_4$
3 $S_4\ q \rightarrow_{\texttt{max}} S_4$
4 $S_4\ f_{j.k} \rightarrow_{\texttt{max}} S_4$
5 $S_4\ c_{j.k} \rightarrow_{\texttt{max}} S_4$
6 $S_4\ l_{j.k} \rightarrow_{\texttt{max}} S_4$
7 $S_4\ t_{j.k} \rightarrow_{\texttt{max}} S_4$
8 $S_4\ t_j \rightarrow_{\texttt{max}} S_4$
9 $S_4\ r_{j.k} \rightarrow_{\texttt{max}} S_4$
10 $S_4\ w \rightarrow_{\texttt{max}} S_4$
11 $S_4\ v \rightarrow_{\texttt{max}} S_4$
12 $S_4\ u \rightarrow_{\texttt{max}} S_4$

13 $S_4\ m \rightarrow_{\texttt{max}} S_4$
14 $S_4\ n \rightarrow_{\texttt{max}} S_4$
15 $S_4\ h \rightarrow_{\texttt{max}} S_4$
16 $S_4\ o \rightarrow_{\texttt{max}} S_4$

5. Rules in state S_5:
   1 $S_5\ c_j p_j \rightarrow_{\texttt{min}} S_5\ p_j$
   2 $S_5\ c_{j.k} \rightarrow_{\texttt{min}} S_5|_{e_j}$
   3 $S_5\ c_{j.k} p_k \rightarrow_{\texttt{min}} S_5\ p_k$
   4 $S_5\ hc_{j.k} \rightarrow_{\texttt{min}} S_5\ hp_k e_j (f_{i.k})_{\updownarrow}|_{\iota_i}$
   5 $S_5\ hc_j \rightarrow_{\texttt{min}} S_5\ hp_j (f_{i.j})_{\updownarrow}|_{\iota_i}$
   6 $S_5\ g \rightarrow_{\texttt{max}} S_4$
   7 $S_5\ b \rightarrow_{\texttt{max}} S_5$
   8 $S_5\ f_{j.k} \rightarrow_{\texttt{max}} S_5$
   9 $S_5\ l_{j.k} \rightarrow_{\texttt{max}} S_5$
   10 $S_5\ t_{j.k} \rightarrow_{\texttt{max}} S_5$
   11 $S_5\ t_j \rightarrow_{\texttt{max}} S_5$
   12 $S_5\ r_{j.k} \rightarrow_{\texttt{max}} S_5$
   13 $S_5\ u \rightarrow_{\texttt{max}} S_5$

When a cell, σ_i, is first reached by a search path, then both its "entry node" and "exit node" become *visited*. If this search path is successful, then σ_i is marked by one object q (rules 3.15, 3.16). In a subsequent round, new search paths can visit σ_i (1) via an incoming arc (forward mode); (2) via an outgoing arc, in the reverse direction (push-back mode) or (3) on both ways. When a search path visits σ_i via an incoming arc, it marks σ_i with one object, m, indicating a visited entry node (rule 1.21); in this case, the search path can only continue with a push-back (rule 2.4). When a search path visits σ_i via an outgoing arc, it marks the cell with one object, n, indicating a visited exit node (rule 1.20); in this case, the search path continues with all other possible arcs (rule 2.5), i.e. all forward searches and also a push-back on its current in-flow arc. A cell which has a visited entry node is in state S_3, but it can be later revisited by its exit node. Thus, in S_3, we provide extra rules to accept and send connect signals (rules 3.3, 3.4).

Cell, σ_i, can be visited at most once on each of its entry or exit nodes; but, it can be visited both on its entry and exit nodes, in which case it has two temporary predecessors (which simulate the node-splitting technique). In Figure 8, the search path, 0.4.5.2.1.8.9.3.2.6.7.10, has visited cell 2 twice, once on its "entry" node and again on its "exit" node. Cell 2 has two temporary predecessors, cells 5 and 3, and receives progress indicators from two successors, cells 1 and 6. Progress indicators relayed by cell 6 must be further relayed to cell 3 and progress indicators relayed by cell 1 must be further relayed to cell 5. To make the right choice, each cell records matching predecessor-successor pairs, e.g., cell 2 records the pairs $t_{5.1}$ and $t_{3.6}$. For example, when the progress indicator $r_{1.2}$ or $f_{1.2}$ arrives, cell 2 knows to forward it to the correct predecessor, cell 5. When the progress indicator $r_{6.2}$ or $f_{6.2}$ arrives, cell 2 knows to forward it to the correct predecessor, cell 3.

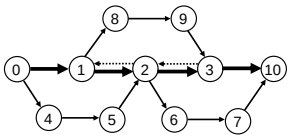


Fig. 8. An example of node-disjoint paths.

The following theorem sums up all these arguments:

Theorem 2. *When Algorithm 2 ends, path predecessors and successors objects mentioned in its output section indicate a maximal cardinality set of node-disjoint paths.*

7 Performance of BFS-based Solutions

Consider a simple P system with n cells, $m = |\delta|$ arcs, where f_e = the maximum number of edge-disjoint paths, f_n = the maximum number of node-disjoint paths and d = the outdegree of the source cell. Dinneen at al. show that the DFS-based algorithms for edge- and node-disjoint paths run in $O(mn)$ P steps [3]. A closer inspection, not detailed here, shows that this upper bound can be improved.

Theorem 3. *The DFS-based algorithms run in $O(md)$ P steps, in both the edge- and node-disjoint cases.*

We show that our algorithms run asymptotically faster ($f_e, f_n \leq d$):

Theorem 4. *Our BFS-based algorithms run in at most $B(m, f) = (3m + 5)f + 4m + 6$ P steps, i.e. $O(mf)$, where $f = f_e$, in the edge-disjoint case, and $f = f_n$, in the node-disjoint case.*

Proof. 1. Initially, the source cell broadcasts a "catalyst" in one step.

2. Then, the algorithm repeatedly searches augmenting paths. First, consider the rounds where augmenting paths are found. In each round, each cell on the search path takes two steps to proceed, i.e. one step to accept a signal and one more step to send connect signals. Each search path spans at most m arcs, thus it takes at most $2m$ steps to reach its end (with or without reaching the target). All search paths in a round proceed in parallel. After the last augmenting path in a round was found, it takes at most m steps to confirm to the source. After receiving the last confirmation signal, the source cell waits four steps (to ensure that it is the last) and then takes one step to broadcast a reset signal. Therefore, each round, where augmenting paths are found, takes at most $3m + 5$ steps. At least one augmenting path is found in each round, so the total number of search rounds is at most f.

3. Next, consider the last search round, where no more augmenting paths are found. This case is similar, but not identical, to the preceding case. Each cell on the search path takes two steps to proceed, so it takes at most $2m$ steps to search augmenting paths. The connect acknowledgment from the end cell of the search path takes at most m steps to arrive at the source. The source waits for three or four steps for the time-out: three steps, if it does not receive any progress indicators; and four steps, otherwise. Then, the source cell broadcast a finalize signal, which takes at most m steps to reach all cells.
4. Finally, all cells take one final step, to clear all irrelevant objects, and the algorithm terminates.

To summarize, the algorithm runs in at most $(3m+5)f+4m+6$ steps and its asymptotic runtime complexity is $O(mf)$.

Table 9 compares the asymptotic complexity of our BFS-based algorithms against some well-known maximum flow BFS-based algorithms. Our BFS-based algorithms are faster, because they leverage the potentially unbounded parallelism inherent in P systems.

Table 9. Asymptotic worst-case complexity: classical BFS-based algorithms (steps), P system DFS-based algorithms [3] (P steps) and our P system BFS-based algorithms (P steps).

Edmonds-Karp [4]	$O(m^2n)$ steps
Dinic [2]	$O(mn^2)$ steps
Goldberg and Tarjan [6]	$O(nm \log n^2/m)$ steps
P System DFS-based [3]	$O(md)$ P steps
P System BFS-based [*here*]	$O(mf)$ P steps

Theorem 4 indicatess the worst-case upper bound, not the typical case. A typical search path does not use all m arcs. Also, the algorithm frequently finds more than one augmenting paths in the same search round, thus the number of rounds is typically much smaller than f. Therefore, the average runtime is probably much less than than the upper bound indicated by Theorem 4. Empirical results, obtained with our in-house simulator (still under development) support this observation.

Table 11, empirically compares the performance of our BFS-based algorithms against the DFS-based algorithms [3], for the scenarios of Figure 10. The empirical results show that BFS-based algorithms take fewer P steps than DFS-based algorithms. The performance is, as expected, influenced by the number of nodes and the density of the digraph. Typically, the ratio of BFS:DFS decreases even more, with the complexity of the digraph. We conclude that, the empirical complexity is substantially smaller than the asymptotic worst-case complexity indicated by Theorem 4.

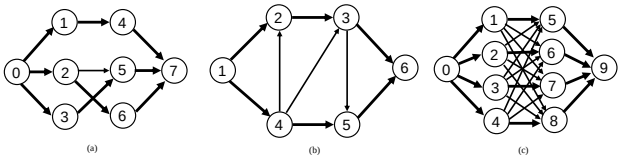


Fig. 10. Empirical tests of BFS-based and DFS-based algorithms.

Table 11. Empirical complexity of BFS-based and DFS-based algorithms (P steps).

Test				BFS Empirical Complexity		DFS Empirical Complexity	
Case	m	$f = f_e, f_n$	$B(m, f)$	Edge-disjoint	Node-disjoint	Edge-disjoint	Node-disjoint
(a)	10	3	151	44	45	63	62
(b)	9	2	106	24	24	61	59
(c)	24	4	410	66	75	241	194

8 Conclusions

We proposed the first BFS-based P system solutions for the edge- and node-disjoint paths problems. As expected, because of potentially unlimited parallelism inherent in P systems, our P system algorithms compare favourably with the traditional BFS-based algorithms. Empirical results show that, in terms of P steps, our BFS-based algorithms outperform the previously introduced DFS-based algorithms [3].

Several interesting questions and directions remain open. Can we solve this problem using a restricted P system without states, without sacrificing the current descriptive and performance complexity? What is the average complexity of our BFS-based algorithms? How much can we speedup the existing DFS-based algorithms, by use more efficient distributed DFS algorithms? An interesting avenue is to investigate a limited BFS design, in fact, a mixed BFS-DFS solution, which combines the advantages of both BFS and DFS. Finally, another direction is to investigate disjoint paths solutions on P systems with asynchronous semantics, where additional speedup is expected.

Acknowledgments

The authors wish to thank Tudor Balanescu, Michael J. Dinneen, Yun-Bum Kim, John Morris and three anonymous reviewers, for valuable comments and feedback that helped us improve the paper.

References

1. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. The MIT Press, 3rd edn. (2009)

2. Dinic, E.A.: Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Math. Dokl., 11, 1277–1280 (1970)
3. Dinneen, M.J., Kim, Y.B., Nicolescu, R.: Edge- and node-disjoint paths in P systems. Electronic Proc. in TCS, 40, 121–141 (2010)
4. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM, 19(2), 248–264 (1972)
5. Ford, L.R., Jr., Fulkerson, D.R.: Maximal flow through a network. Canadian Journal of Mathematics, 8, 399–404 (1956)
6. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. J. ACM, 35(4), 921–940 (1988)
7. Ionescu, M., Sburlan, D.: On p systems with promoters/inhibitors. J. Universal Computer Science, 10(5), 581–599 (2004)
8. Kozen, D.C.: The Design and Analysis of Algorithms. Springer, New York, NY, USA (1991)
9. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
10. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theor. Comput. Sci. 296(2), 295–326 (2003)
11. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Structured modelling with hyperdag P systems: Part A. Report CDMTCS-342, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand (December 2008), http://www.cs.auckland.ac.nz/CDMTCS/researchreports/342hyperdagA.pdf
12. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Structured modelling with hyperdag P systems: Part B. Report CDMTCS-373, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand (October 2009), http://www.cs.auckland.ac.nz/CDMTCS//researchreports/373hP_B.pdf
13. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Towards structured modelling with hyperdag P systems. Intern. J. Computers, Comm. and Control, 2, 209–222 (2010)
14. Păun, G.: Membrane Computing: An Introduction. Springer, New York, Inc., Secaucus, NJ, USA (2002)
15. Păun, G.: Introduction to membrane computing. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.) Applications of Membrane Computing, pp. 1–42. Natural Computing Series, Springer (2006)
16. Păun, G., Centre, T., Science, C.: Computing with membranes. J. Computer and System Sciences, 61, 108–143 (1998)
17. Seo, D., Thottethodi, M.: Disjoint-path routing: Efficient communication for streaming applications. In: IPDPS. pp. 1–12. IEEE (2009)
18. Tel, G.: Introduction to Distributed Algorithms. Cambridge Univ. Press (2000)

On a Contribution of Membrane Computing to a Cultural Synthesis of Computer Science, Mathematics, and Biological Sciences

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences,
Śniadeckich 8, P.O.Box 21, 00-956 Warsaw, Poland
e-mail: adamo@impan.gov.pl

Summary. Some topic contribution of membrane computing to a cultural synthesis of computer science, mathematics and biological sciences is presented.

1 Introduction

After a more than decade of the researches in the area of membrane computing, cf. [19], initiated by Gheorghe Păun it is worth to propose a discussion about a contribution of these researches to a cultural synthesis of computer science, mathematics, and biological sciences.

An aim of the present paper is to initiate the discussion focusing on these its aspects which are familiar to the author whose research area comprises mathematical foundations of computer science and foundations of mathematics itself.

The theme of the discussion was inspired by [11] and [2], where the goal of [11] is to propose a framework that should allow a healthy and positive role for speculations in mathematics although traditional mathematical norms discourage speculation whereas it is the fabric of theoretical physics. Thus a cultural synthesis of mathematics and theoretical physics is understood in [11] more or less explicitly as a mutual inspiration between the discussed areas.

We point out here that speculation in mathematics can be reinforced by computer experiments, cf. provocative article [10], which could extend the discussion in [11] to computer science with a regard to the relations between mathematical thinking and algorithmic thinking, cf. [12].

The responses in [2] contain various opinions about a role of speculation in mathematics from a defense of an importance of proofs, see S. Mac Lane's response continued in [14], to an acceptance of speculation as possessing equal rights to proofs. The response due to R. Thom in [2] suggests also a cultural synthesis of mathematics and biology.

We propose a looking forward approach to a cultural synthesis of the areas mentioned in the title of the present paper, where a cultural synthesis is also understood as mutual inspiration.

In the next section we outline some topic contribution of membrane computing to this cultural synthesis.

2 Topic contribution

We point out the following topics as a contribution of membrane computing to the discussed synthesis:

A) biologically inspired self-assembly (randomized, cf. [15] and [16]) P systems (with membrane division and creation) for solving NP complete problems, cf. [17] and [21], which extend algorithmic thinking by a new idea of a (randomized) algorithm for a construction of a self-assembly distributed system realizing massively parallel computation,
B) spiking neural P systems, cf. [18], with learning problem solution, cf. [9], which are a step towards digitalization[1] of neural networks within mathematical neuroscience, cf. e.g. [3],
C) fractal constructs generated by P systems, cf. [8], with P systems for obtaining homology groups, cf. [5], and a possibility to experiment with them (to predict some mathematical results like in [10] to be proved with mathematical precision like in [14]) via P-lingua programming environment designed in Seville, cf. [4] and [20].

Concerning A) one can ask for a relation of randomized P systems in [15] and randomized Gandy–Păun–Rozenberg machines [16] with probabilistic computing devices discussed in [1].

References

1. Arora, S., Barak, B., *Computational Complexity: A Modern Approach*, Cambridge Univ. Press, Cambridge, 2009.
2. Atiyah, M., et al., *Responses to* [11], Bull. Amer. Math. Soc. 30 (1994), pp. 178–207.
3. Bohte, S. M., *Spiking Neural Networks*, Professorschrift, Leiden University, 2003.
4. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M. J., Riscos-Núñez, A., *A P-Lingua Programming Environment for Membrane Computing*, in: Membrane Computing, Lecture Notes in Computer Science 5391, Springer, Berlin, 2009, pp. 187–203.
5. Díaz-Pernil, D., Gutiérrez-Naranjo, M. A., Real, P., Sanchez-Canales, V., *A cellular way to obtain homology groups in binary 2D images*, in: Eight Brainstorming Week on Membrane Computing, Seville 2010, ed. M. A. Martinez-del-Amor et al., RGNC Report 01/2010, Seville University, pp. 89–99.

[1] in a similar way as cellular automata digitalize physics according to E. Fredkin, S. Wolfram, cf. [6], [7], [13]

6. Fredkin, E., *An introduction to digital philosophy*, Internat. J. Theor. Phys. 42 (2003), pp. 189–247.
7. Fredkin, E., *Digital mechanics*, Phys. D 45 (1990), pp. 254–270.
8. Gutiérrez-Naranjo, M. A., Pérez-Jiménez, M. J., *Fractals and P systems*, in: Fourth Brainstorming Week on Membrane Computing, Seville 2006, vol. II, ed. C. Graciani et al., Fenix Editora, 2006, pp. 65–86.
9. Gutiérrez-Naranjo, M. A., Pérez-Jiménez, M. J., *A spiking neural P systems based model for Hebbian learning*, in: Proceedings of 9th Workshop on Membrane Computing, Edinburgh, July 28 – July 31, 2008, ed. P. Frisco et al., Technical Report HW-MASC-TR-0061, School of Mathematical and Computer Sciences, Heriot–Watt University, Edinburgh, UK, 2008, pp. 189–207.
10. Horgan, J., *The death of proof*, Scientific American 269, no. 4 (October 1993), pp. 74–82.
11. Jaffe, A., Quinn, F., *Theoretical mathematics: Towards a cultural synthesis of mathematics and theoretical physics*, Bull. Amer. Math. Soc. 29 (1993), pp. 1–13,
12. Knuth, D. E., *Algorithmic thinking and mathematical thinking*, Amer. Math. Monthly 92 (1985), pp. 170–181.
13. Kůrka, P., *Topological and Symbolic Dynamics*, Société Mathématique de France, Paris, 2003.
14. S. Mac Lane, *Despite physicists, proof is essential in mathematics*, Synthese 111 (1997), pp. 147–154.
15. A. Obtułowicz, *Probabilistic P systems*, in: Membrane Computing, ed. Gh. Păun et al., Lecture Notes in Computer Science 2597, Springer, Berlin, 2003, pp. 377–387.
16. A. Obtułowicz, *Randomized Gandy–Păun–Rozenberg machines*, in: Membrane Computing, Lecture Notes in Computer Science 6501, Springer, Berlin, 2011, pp. 305–324.
17. Păun, Gh., *P systems with active membranes: Attacking NP complete problems*, Journal of Automata, Languages and Combinatorics 6 (2000), pp. 75–90.
18. Păun, Gh., Pérez-Jiménez, M. J., *Spiking neural P systems. Recent results, research topics*, presented at the 6th Brainstorming Week on Membrane Computing, Sevilla 2008, web page `http://psystems.disco.unimib.it/download/leidenGR65.pdf`
19. Păun, Gh., Rozenberg, G., Salomaa, A., *The Oxford Handbook of Membrane Computing*, Oxford, 2009.
20. The P-Lingua website, `http://www.p-lingua.org`
21. Zandron, C., Ferretti, C., Mauri, G., *Solving NP complete problems with active membranes*, in: Unconventional Models of Computation, UMC'2K, ed. I. Antoniu et al., Berlin, 2001, pp. 289–301.

Well-Tempered P Systems: Towards a Membrane Computing Environment for Music Composition

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences,
Śniadeckich 8, P.O.Box 21, 00-956 Warsaw, Poland
e-mail: adamo@impan.gov.pl

Summary. A proposal of designing a membrane computing environment for music composition is outlined.

1 Introduction

We outline a proposal of designing a membrane computing environment for music composition, where a basic theoretical concept for this environment is a notion of a well-tempered P system with the adjective "well-tempered" understood here in a similar way as in the title of *Well-tempered computer* of [12] and with the notion of a P system defined as in [9]. The notion of a well-tempered P system is explained in the next section as a result of a synthesis of some known concepts.

2 Well-tempered P systems

The notion of a well-tempered P system is proposed to be a result of synthesis of the following known concepts, ideas, and notions:

— the notion of a λP system due to N. Jonoska and M. Margenstern, cf. [1] and [6],
— a concept of a music score, written or analyzed, in terms of music calculi having common features with (type-free) lambda calculus, cf. [8] and [7],
— the idea of sounding P system, cf. [3],

with a regard to M. Steedman's categorical grammar approach to jazz improvisation, cf. [13].

More precisely, a well-tempered P system is aimed to represent a score of a music piece, e,g. fuge, written in terms of a music calculus, cf. [8] and [7], like λP systems represent λ terms, respectively, where a reduction process of a λ term

corresponds to an evolution process generated by the lambda P system representing this λ term, cf. [1] and [6].

We do not exclude other approaches to the notion of a well-tempered P system, where:

— a P system for modelling higher plants, cf. [10], could represent a score of music piece expanding in time like plants with some probability factor like in [5],
— a P system for fractal generation, cf. [4], could represent a score of music piece expanding in time like cellular automata generating fractals, cf. [11].

Therefore the P-Lingua tools, cf. [2] and [14], could provide an appropriate software for an environment for music composition designed in the frames of membrane computing by using well-tempered P systems.

References

1. Colson, L., Jonoska, N., Margenstern, M., *λP systems and typed λ-calculus*, in: Membrane Computing, LNCS 3365, Springer, Berlin, 2005, pp. 1–18.
2. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M. J., Riscos-Núñez, A., *A P-Lingua Programming Environment for Membrane Computing*, in: Membrane Computing, LNCS 5391, Springer, Berlin, 2009, pp. 187–203.
3. Garcia-Quasimodo, M., Gutiérrez-Naranjo, M. A., Ramírez-Martínez, D., *How does a P system sound?*, in: Eight Brainstorming Week on Membrane Computing, Seville 2010, ed. M.A. Martinez-del-Amor et al., RGNC Report 01/2010, Seville University, pp. 123–132.
4. Gutiérrez-Naranjo, M. A., Pérez-Jiménez, M. J., *Fractals and P systems*, in: Fourth Brainstorming Week on Membrane Computing, Seville 2006, vol. II, ed. C. Graciani et al., Fenix Editora, 2006, pp. 65–86.
5. Hiller, L. A., Isaacson, L. M., *Experimental Music Composition with Electronic Computer*, McGraw–Hill, 1959.
6. Jonoska, N., Margenstern, M., *Tree operators in P systems and λ-calculus*, Fundamenta Informaticae 59 (2004), pp. 67–90.
7. Letz, S., Faber, D., Orlarey, Y., *The role of lambda abstraction in elody*, in: Proceedings ICMC'98, 1998.
8. Orlarey, Y., Faber, D., Letz, S., Bilton, M. *Lambda calculus and music calculi*, in: Proceedings ICMC'94, San Francisco, 1994.
9. Păun, Gh., Rozenberg, G., Salomaa, A., *The Oxford Handbook of Membrane Computing*, Oxford, 2009.
10. Romero-Jiménez, A., Gutiérrez-Naranjo, M. A., Pérez-Jiménez, M. J., *Graphical modelling of higher plants using P systems*, in: Membrane Computing, LNCS 4361, Springer, Berlin, 2006, pp. 496–506.
11. Solomos, M., *Cellular automata in Xenakis music. Theory and practice*, in: Proceedings of the International Symposium Iannis Xenakis (Athens, May 2005), ed. A. Georgaki and G. Zervos, pp. 120–138.
12. Steedman, M., *The well-tempered computer*, Philosophical Transactions: Physical Sciences and Engineering 349 (1994), no. 1689, pp. 115–131.
13. Steedman, M., *The blues and the abstract truth: music and mental models*, in: Mental Models in Cognitive Science, ed. A. Garnham and J. Oakhill, 1996.
14. The P-Lingua website, `http://www.p-lingua.org`

dP Automata versus Right-Linear Simple Matrix Grammars

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

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es

Summary. We consider dP automata with the input string distributed in an arbitrary (hence not necessary balanced) way, and we investigate their language accepting power, both in the case when a bound there is on the number of objects present inside the system and in the general case. The relation with right-linear simple matrix grammars is useful in this respect. Some research topics and open problems are also formulated.

1 Introduction

dP automata are a class of computing devices considered in membrane computing area in order to have a distributed language accepting machinery, with the strings to recognize being split among the components of the system and with these components working in parallel on the input strings. In the general case, dP systems consist of a given number of components in the form of a usual symport/antiport P system, which can have their separate inputs and communicate from skin to skin membranes by means of antiport rules like in tissue-like P systems. Such devices were introduced in [7] and further investigated in [3], [8], [9], mainly comparing their power with that of usual P automata and with families of languages in the Chomsky hierarchy. In the basic definition and in all these papers, following the style of the communication complexity area (see, [4]), the so-called *balanced* mode of introducing the input string is considered: the string is split in equal parts, modulo one symbol, and distributed among components.

Here we consider the general case, with no restriction on the input string distribution; each component just takes symbols from the environment when it can do it it, without any restriction on their number. This is a very natural and general set-up, which, however, was only incidentally investigated so far. Two cases are distinguished: with a bound on the size of the system (on the total number of objects present inside) and without such a bound. Both cases are naturally related to

a classic family of regulated grammars, the simple matrix grammars of [5] (see also [2]). Actually, as expected, right-linear simple matrix grammars are closely related to dP automata, and we will examine below this connection (looking for mutual simulations among the two types of language identifying machineries). This connection was already pointed out in [8], where the conjecture was formulated that, in the same way as a usual finite automaton can be simulated by a P automaton, a right-linear simple matrix grammar can be simulated by a dP automaton. We confirm here this conjecture (in the general, not the balanced case).

2 Formal Language Theory Prerequisites

The reader is assumed to have some familiarity with basics of membrane computing, e.g., from [6], [10], and of formal language theory, e.g., from [2], [11], but we recall below all notions necessary in the subsequent sections.

In what follows, V^* is the free monoid generated by the alphabet V, λ is the empty word, $V^+ = V^* - \{\lambda\}$, and $|x|$ denotes the length of the string $x \in V^*$. REG, LIN, CF, CS, RE denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively.

Essential below will be the right-linear simple matrix grammars introduced in [5]. Such a grammar of degree $n \geq 1$ is a construct of the form $G = (N_1, \ldots, N_n, T, S, M)$, where $N_1, N_2, \ldots, N_n, T$ are pairwise disjoint alphabets (we denote by N the union of $N_1, \ldots, N_n$), $S \notin T \cup N$, and M contains matrices of the following forms:

(i) $(S \to x), x \in T^*$,
(ii) $(S \to A_1 A_2 \ldots A_n),\ A_i \in N_i, 1 \leq i \leq n$,
(iii) $(A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n),\ A_i, B_i \in N_i, x_i \in T^*, 1 \leq i \leq n$,
(iv) $(A_1 \to x_1, \ldots, A_n \to x_n),\ A_i \in N_i, x_i \in T^*, 1 \leq i \leq n$.

A derivation starting with a matrix of type (ii) continues with an arbitrary numbers of steps which use matrices of type (iii) and ends by applying a matrix of type (iv).

We denote by $L(G)$ the language generated in this way by G and by RSM_n the family of languages $L(G)$ for right-linear simple matrix grammars G of degree at most n, for $n \geq 1$. The union of all these families is denoted by RSM_*. The strict inclusions $RSM_n \subset RSM_{n+1}, n \geq 1$, are known. Moreover, $REG = RSM_1$, $RSM_* \subset CS$, RSM_* is incomparable with LIN and CF, all languages in RSM_* are semilinear, and this family is closed under union, intersection with regular languages, direct and inverse morphisms (but not under intersection, complement and Kleene +).

Clearly, a normal form can be easily found for these grammars: in matrices of type (iii) we can ask to have $x_i \in T \cup \{\lambda\}, 1 \leq i \leq n$, and in matrices of type (iv) we can have $x_i = \lambda$ for all $1 \leq i \leq n$.

3 dP Automata

We introduce now the computing devices we investigate in this paper, also giving a relevant example.

As usual in membrane computing, the multisets over an alphabet V are represented by strings in V^*; a string and all its permutations correspond to the same multiset, with the number of occurrences of a symbol in a string representing the multiplicity of that object in the multiset. (We work here only with multisets of finite multiplicity.) The terms "symbol" and "object" are used interchangeably, all objects are here represented by symbols.

A *dP automaton* (of degree $n \geq 1$) is a construct

$$\Delta = (O, E, \Pi_1, \ldots, \Pi_n, R),$$

where:

(1) O is an alphabet (of objects);
(2) $E \subseteq O$ (the objects available in arbitrarily many copies in the environment);
(3) $\Pi_i = (O, \mu_i, w_{i,1}, \ldots, w_{i,k_i}, E, R_{i,1}, \ldots, R_{i,k_i})$ is a symport/antiport P system of degree k_i (O is the alphabet of objects, μ_i is a membrane structure of degree k_i, $w_{i,1}, \ldots, w_{i,k_i}$ are the multisets of objects present in the membranes of μ_i in the beginning of the computation, E is the alphabet of objects present – in arbitrarily many copies – in the environment, and $R_{i,1}, \ldots, R_{i,k_i}$ are finite sets of symport/antiport rules associated with the membranes of μ_i; the symport rules are of the form $(u, in), (u, out)$, where $u \in O^*$, and the antiport rules are of the form $(u, out; v, in)$, where $u, v \in O^*$; note that we do not have an output membrane), with the skin membrane labeled with $(i, 1) = s_i$, for all $i = 1, 2, \ldots, n$;
(4) R is a finite set of rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n, i \neq j$, and $u, v \in O^*, uv \neq \lambda$.

The systems $\Pi_1, \ldots, \Pi_n$ are called *components* of Δ and the rules in R are called *communication rules*. For a rule $(s_i, u/v, s_j)$, $|uv|$ is the *weight* of this rule.

Using a rule $(u, in), (u, out)$ associated with a membrane i means to bring in the membrane, respectively to send out of it the multiset u; using a rule $(u, out; v, in)$ associated with a membrane i means to send out of the membrane the objects of multiset u and, simultaneously, to bring in the membrane, from the region surrounding membrane i, the objects of multiset v. A communication rule $(s_i, u/v, s_j)$ moves the objects of u from component Π_i to component Π_j, simultaneously with moving the objects in the multiset v in the opposite direction.

Each component Π_i can take symbols from the environment, work on them by using the rules in sets $R_{i,1}, \ldots, R_{i,k_i}$, and communicate with other components by means of rules in R.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over O if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, using the symport/antiport rules as well as the inter-components communication

rules, in the non-deterministic maximally parallel way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts. A problem appears in the case when several objects are read at the same time from the environment, by several rules or by a single rule of the form $(u, out; v, in)$, with $|v| \geq 2$; in such a case any permutation of the symbols brought in the system in the same step are considered as a valid substring of the input string (thus, a computation can recognize several strings, differing to each other by permutations of certain substrings). Note that we impose here no condition on the relative lengths of strings $x_1, x_2, \ldots, x_n$ (as it is done in previous papers dealing with dP automata, under the influence of communication complexity area). We denote by $L(\Delta)$ the language of all strings recognized by Δ in this way, and by LdP_n the family of languages $L(\Delta)$, for Δ of degree at most $n \geq 1$. The union of all these families is denoted by LdP_*.

The dP automata are synchronized devices, a universal clock exists for all components, marking the time in the same way for the whole dP automaton. When the system has only one component, then we obtain the usual notion of a P automaton, as investigated in a series of papers (mainly in the extended version, with a terminal alphabet of objects – see the respective chapter in [10] and the references therein). We denote by LP the family of languages recognized by P automata. Hence, $LP = LdP_1$ and, from [3], it is known that $REG \subset LP \subset CS$ and LP is incomparable with CF.

We consider now a somewhat surprising example, of a dP automaton of degree 2, generating a complex language, $L_1 = \{ww \mid w \in \{a, b\}^*\}$. The automaton is given in Figure 1, in the standard way of representing a dP automaton. We have $O = \{a, b, c_1, c_2, d, \#\}$ and $E = \{a, b\}$.

All antiport rules which bring objects from the environment are of weight one, hence the number of objects present in the system is constant, four in each component. In the first step, objects d release $c_2 a$ in the skin region of the first component and $c_1 a$ in the second. Each symbol a can bring either an a or a b from the environment and, at the same time, the objects c_1, c_1 are interchanged between the two components (otherwise, they release the trap object #, which will oscillate forever across membranes $(1, 1)$, respectively, $(2, 1)$, and the computation never stops). With $c_1\alpha, \alpha \in \{a, b\}$, in the first component and $c_2\beta, \beta \in \{a, b\}$, in the second one, the only continuation which does not release the trap object is possible when $\alpha = \beta$, by using the communication rule $(s_1, c_1\alpha / c_2\alpha, s_2)$ (if one of the symbols α, β brings new symbols from the environment, the corresponding c_1, c_2 should enter the membrane $(1, 2)$ or $(2, 2)$, bringing out the object #). We obtain a configuration as that we started with, hence the process can be iterated. If, at any moment when c_2 is in Π_1 and c_1 is in Π_2, one of the rules $(c_2\alpha, in), \alpha \in \{a, b\}$, is used in the first component, or $(c_1\alpha, in), \alpha \in \{a, b\}$, is used in the second component, then this should be done simultaneously in both components, otherwise again one of c_1, c_2 has to release the trap object. In conclusion, the strings read from the environment by the two components are identical, hence $L(\Delta) = L_1$.

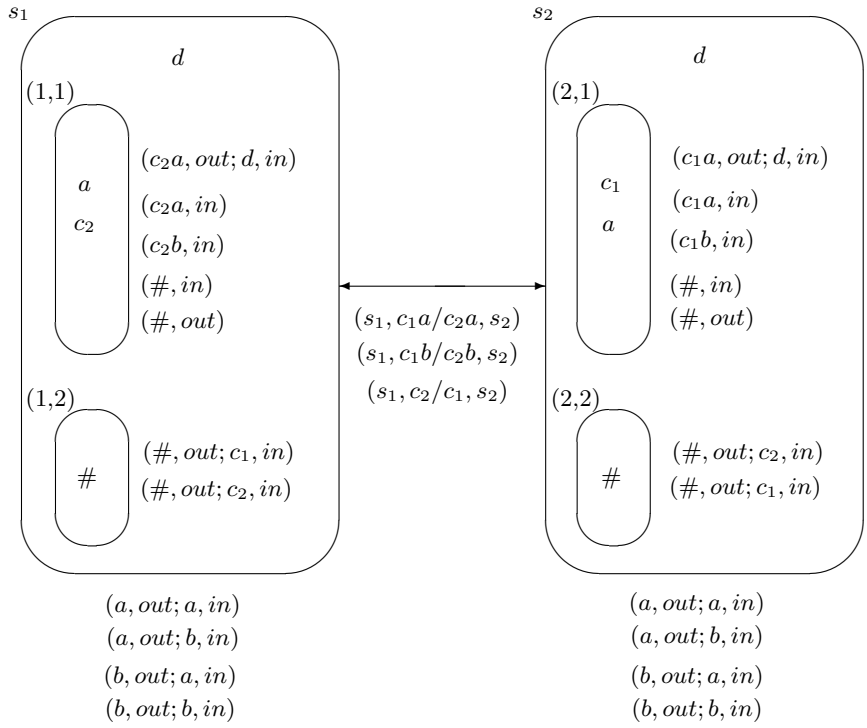


Fig. 1. A dP automaton recognizing the language L_1.

Note the important facts that the system reads the input in a balanced way and that it is *bounded*, the total number of objects present inside is always bounded by a constant (8 in our case) given in advance. This last characteristics is important, so that we denote by $LdP^b_n, n \geq 1$, the family of languages recognized by bounded dP automata of degree at most n; when n is not specified, we replace it by $*$.

4 The Power of dP Automata

We start by reformulating in a more general way a result already suggested by a proof in [9].

Theorem 1. *$LdP^b_n \subseteq RSM_n$, for all $n \geq 1$.*

Proof. Let Δ be a dP automaton of degree n (with the set of objects O) which is bounded. Then, the set of all its configurations is finite. Let $\sigma_0, \sigma_1, \ldots, \sigma_p$ be this set, with σ_0 being the initial configuration. We construct the following right-linear simple matrix grammar:

$$
\begin{aligned}
&G = (N_1, \ldots, N_n, O, S, M), \text{ with}\\
&N_i = \{(\sigma_j)_i \mid 0 \le j \le p\},\ i = 1, 2, \ldots, n,\\
&M = \{(S \to (\sigma_0)_1(\sigma_0)_2 \ldots (\sigma_0)_n)\}\\
&\quad \cup \{(\sigma_i)_1 \to \alpha_1(\sigma_j)_1, \ldots, (\sigma_i)_n \to \alpha_n(\sigma_j)_n) \mid\\
&\qquad \text{from configuration } \sigma_i \text{ the dP automaton } \Delta \text{ can pass to}\\
&\qquad \text{the configuration } \sigma_j \text{ by a correct transition, taking from the}\\
&\qquad \text{environment the objects } \alpha_1, \ldots, \alpha_n \text{ by its components, where}\\
&\qquad \alpha_s \in O \cup \{\lambda\}, 1 \le s \le n\}\\
&\quad \cup \{(\sigma_h)_1 \to \lambda, \ldots, (\sigma_h)_n \to \lambda) \mid \sigma_h \text{ is a halting configuration}\}.
\end{aligned}
$$

Note that all nonterminals in the rules of a matrix contain the same "core information", namely the current configuration of the system, hence the complete control of the system working is obtained in this way. The equality $L(\Delta) = L(G)$ is obvious. □

This result cannot be extended to arbitrary dP automata. Actually, we have:

Theorem 2. $LdP_2 - RSM_* \neq \emptyset$.

Proof. Let us consider the following dP automaton:

$$
\begin{aligned}
&\Delta = (O, E, \Pi_1, \Pi_2, R), \text{ with}\\
&O = \{a, c, d, e, f, \#\},\\
&E = \{a, c, d, e\},\\
&\Pi_1 = (O, [\ \ ]_{s_1}, f, E, \{(f, out; a, in),\ (a, out; aa, in)\}),\\
&\Pi_2 = (O, [\,[\ \ ]_{(2,1)}\,]_{s_2}, E, \{(f, out; d, in),\ (a, out; c, in),\ (d, out; e, in)\},\\
&\qquad\qquad \{(f, out; f, in)\}),\\
&R = \{(s_1, a/\lambda, s_2)\}.
\end{aligned}
$$

For an easier examination of the work of the system, we also represent it graphically, in Figure 2.

Let us look for strings accepted by this dP automaton which are of the form $a^i dc^j e$, for some $i, j \ge 1$.

After introducing the symbol a in the first component, let us assume that for $n \ge 0$ steps we use here the rule $(a, out; aa, in)$, hence we produce 2^n copies of a in Π_1, while the second component uses the rule $(f, out; f, in) \in R_{(2,1)}$. Suppose now that $p \ge 0$ copies of a remains in the first component and the others $r = 2^n - p$ are moved to the second component. Here, all r copies of a must go out, in exchange of objects c, hence the string read by the second component starts with c^r. At the same time or one step before, the second component must introduce the symbol d. This object becomes immediately e, hence the exchange of a for c should be done either in the same step with reading d or at the same time with reading e in

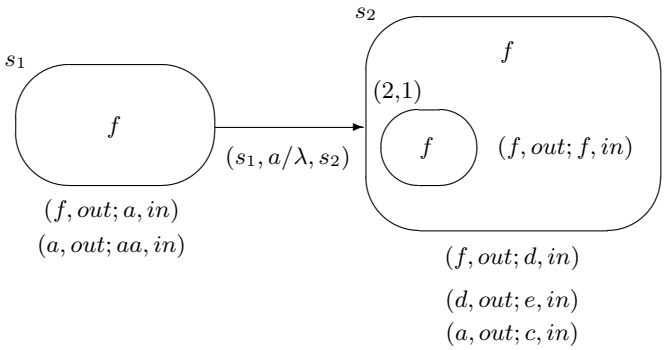


Fig. 2. A dP system recognizing a language not in RSM_*

the second component (because any permutation of the objects is allowed in the string, either variant is possible). However, after e, we do not want to have any symbol, hence all copies of a were already moved to the second component, and thus the work of the first component stops. When introducing the symbol d in the second component, the p copies of a from the first component cannot use the rule $(a, out; aa, in)$, but they must come immediately in the second component, to introduce c here at the same time with introducing e. Therefore, if the string has the form $a^i dc^j e$, then $i = j = 2^n$ for some $n \geq 0$ ($n = 0$ is obtained if the unique a introduced in the first step in Π_1 is immediately sent to component Π_2).

Consequently, $L(\Delta) \cap a^*dc^*e = \{a^{2n}dc^{2n}e \mid n \geq 0\}$, which is not in RSM_*, hence also $L(\Delta)$ is not in RSM_*: this family is closed under intersection with regular languages and contains only semilinear languages. □

Note that the previous construction takes the input string in an almost balanced way, and, if in the first step, the first component uses a rule $(f, out; dea, in)$ instead of $(f, out; a, in)$, then we have a balanced functioning, hence the result in the previous theorem holds true also for the balanced way of defining the recognized string.

We pass now to the counterpart of Theorem 1 announced above.

Theorem 3. *$RSM_n \subseteq LdP^b_{n+1}$, for all $n \geq 1$.*

Proof. Let us consider a right-linear simple matrix grammar $G = (N_1, \ldots, N_n, T, S, M)$ as introduced in Section 2, with the alphabets $N_1, N_2, \ldots, N_n$ (their union is denoted by N) and T. Matrices of the form (i), $(S \to x), x \in T^*$, can be replaced by matrices of forms (ii), (iii) and (iv), in an obvious way, hence we assume that we do not have such matrices. We assume all matrices labeled in a one-to-one way; let $m_j : (A_1 \to x_1B_1, \ldots, A_n \to x_nB_n)$, with $1 \leq j \leq k$, be all matrices of type (iii), with $A_i, B_i \in N_i, x_i \in T^*, 1 \leq i \leq n$. Similarly, let $m_j : (A_1 \to x_1, \ldots, A_n \to x_n)$, with $k+1 \leq j \leq p$, be all matrices of

type (iv), with $A_i \in N_i, x_i \in T^*, 1 \leq i \leq n$. Without any loss of the generality we can assume that all strings x_i in these matrices are from $T \cup \{\lambda\}$.

For each matrix, of any form, $m_j : (A_1 \to u_1, \ldots, A_i \to u_i, \ldots, A_n \to u_n)$, let us consider the symbol $[m_j, A_i \to u_i]$ (thus identifying the matrix and its ith rule), and let $X_j(i)$ be a shorthand for it. Consider the alphabets

$$M_i = \{X_j(i) \mid 1 \leq j \leq p\}, \text{ for all } 1 \leq i \leq n.$$

We also denote by M_i' the alphabet of primed symbols in M_i.

For a matrix $m_j : (A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n)$ of type (iii), let us denote $lhs_j = A_1 A_2 \ldots A_n$ and $rhs_j = B_1 B_2 \ldots B_n$. Similarly, for a matrix $m_j : (A_1 \to x_1, \ldots, A_n \to x_n)$ of type (iv), we denote $lhs_j = A_1 A_2 \ldots A_n$.

If $rhs_j = lhs_k$, then we write $m_j \rightsquigarrow m_k$. Similarly, we write $S \rightsquigarrow m_j$ if $(S \to A_1 A_2 \ldots A_n) \in M$ and $A_1 A_2 \ldots A_n = lhs_j$.

For a set Q, we denote by Q also the multiset consisting of the elements of Q, with the multiplicity one for each of them (hence Q can be considered also as the string composed by the elements of the set, in any ordering).

We are now ready to construct the dP system we look for (a_0 is an arbitrary symbol of T fixed in advance):

$$\begin{aligned}
\Delta &= (O, E, \Pi_1, \ldots, \Pi_{n+1}, R), \text{ with :}\\
O &= \bigcup_{i=1}^{n} (M_i \cup M_i') \cup T \cup \{c_i \mid 1 \leq i \leq n\} \cup \{d, f, \#\},\\
E &= T,\\
\Pi_i &= (O, [\,[\quad]_{(i,1)}[\quad]_{(i,2)}\,]_{s_i}, \lambda, M_i' T c_i, \#, R_{s_i}, R_{(i,1)}, R_{(i,2)}),\\
&\quad R_{s_i} = \{(a, out; b, in) \mid a, b \in T\},\\
&\quad R_{(i,1)} = \{(X_j'(i), out; X_j(i), in),\\
&\quad (X_j(i) c_i a, out; X_j'(i) c_i a, in) \mid 1 \leq j \leq p,\\
&\quad \text{ if } X_j(i) = [m_j, A_i \to a B_i], a \in T\}\\
&\quad \cup \{(X_j'(i) a, out; X_j(i) a, in),\\
&\quad (X_j(i) c_i a, out; X_j'(i) c_i a, in) \mid 1 \leq j \leq p, a \in T,\\
&\quad \text{ if } X_j(i) = [m_j, A_i \to B_i]\}\\
&\quad \cup \{(\#, in),\ (\#, out)\},\\
&\quad R_{(i,2)} = \{(\#, out; c_i, in)\}\\
&\quad \cup \{(\#, out; X_j(i), in) \mid 1 \leq j \leq p, \text{ if } X_j(i) = [m_j, A_i \to B_j]\},\\
&\quad \text{ for all } 1 \leq i \leq n,\\
\Pi_{n+1} &= (O, [\,[\ ]_{(n+1,1)}\,]_{s_{n+1}}, c_1 \ldots c_n f, M_1^2 \ldots M_n^2 T^n a_0^n, \emptyset, R_{(n+1,1)}),\\
&\quad R_{(n+1,1)} = \{(X_j(1) \ldots X_j(n) a_0^n, out; f, in) \mid 1 \leq j \leq p \text{ if } S \rightsquigarrow m_j\}\\
&\quad \cup \{(X_k(1) a_1 \ldots X_k(n) a_n, out; X_j(1) a_1 \ldots X_j(n) a_n, in)\\
&\quad \mid 1 \leq j, k \leq p, a_i \in T, 1 \leq i \leq n, \text{ if } m_j \rightsquigarrow m_k\}
\end{aligned}$$

$$\cup \{(X_j(1)c_1a_1 \dots X_j(n)c_na_n, in) \mid a_i \in T, 1 \le i \le n, \text{ if } m_j \text{ is a terminal matrix}\},$$
$$R = \{(s_i, \lambda/c_i, s_{n+1}),$$
$$(s_i, c_i/X_j(i)a, s_{n+1},$$
$$(s_i, X_j(i)c_ia/\lambda, s_{n+1}) \mid 1 \le j \le p, 1 \le i \le n, a \in T\}.$$

This dP system, with one component Π_i and with Π_{n+1} given in full details, is represented in Figure 3.

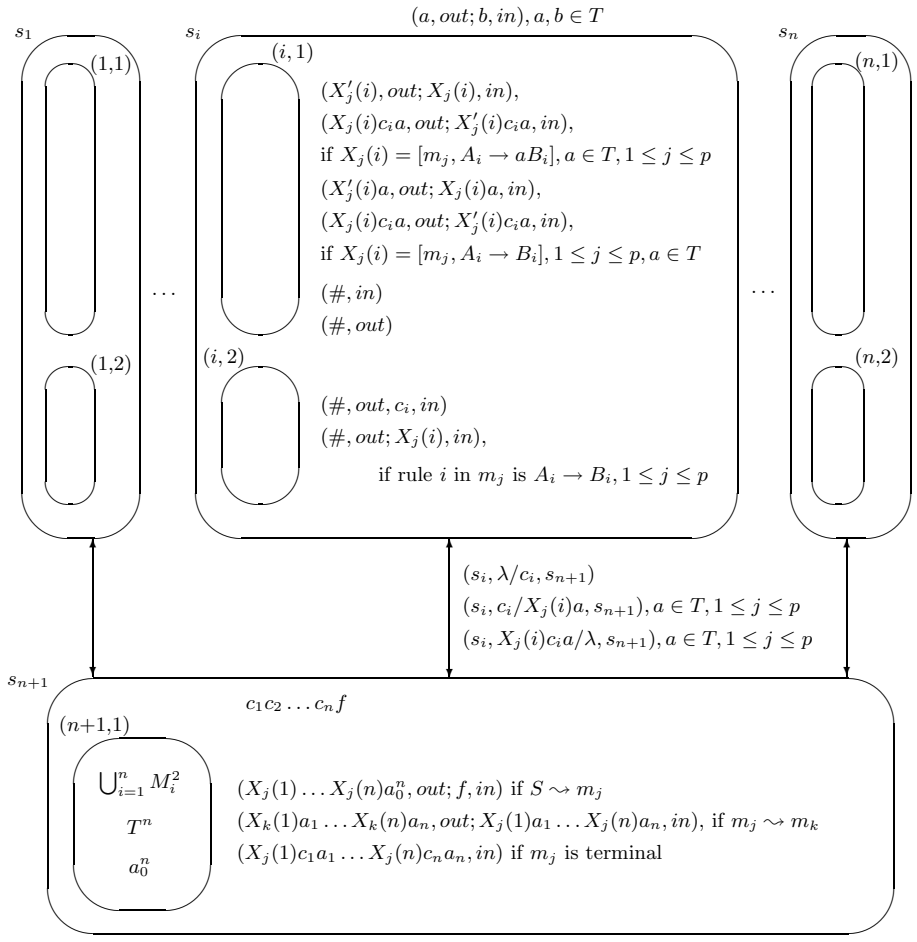


Fig. 3. The dP system in the proof of Theorem 3

The components $\Pi_i, 1 \le i \le n$, simulate the corresponding "component" of the grammar G, while Π_{n+1} is a "synchronizer" of the other components, it takes no

objects from the environment. All rules which bring objects from the environment are uniport rules, hence the system is bounded, the number of objects inside it remains constant during the computation.

We start by sending objects c_i from Π_{n+1} to components Π_i, simultaneously releasing from membrane $(n+1,1)$ some objects $X_j(i), 1 \leq i \leq n$, for a matrix m_j which can follow immediately after an initial matrix of G; each symbol $X_j(i)$ is accompanied by a copy of the symbol a_0, arbitrarily chosen from T.

In the next step, we have to exchange the symbol c_i from Π_i with $X_j(i)a_0$ from Π_{n+1} (if c_i remains unused in Π_i, then it will release the trap object # from membrane $(i,2)$, and the computation will never halt).

In the next step, c_i comes back to Π_i, and in this component we have two possibilities:

(1) The rule i from m_j is of the form $A_i \to aB_i$, and then we use a rule $(a_0, out; b, in)$, for some $b \in T$, and $(X'_j(i), out; X_j(i), in)$.

Now, we check whether the simulation of the rule in G is correct (hence b was the right symbol to take from the environment, i.e., $a = b$): c_i cannot return to Π_{n+1} alone and cannot stay unused in Π_i. The only continuation which does not lead to an infinite computation is to use the rule $(X_j(i)c_ia, out; X'_j(i)c_ia, in)$. These three objects, $X_j(i)c_ia$, can now move together to Π_{n+1}. The only continuation is to move again c_i in Π_i, for all i, and to exchange $X_j(1)\dots X_j(n)$ for some $X_k(1)\dots X_k(n)$ in Π_{n+1}, for $m_j \rightsquigarrow m_k$.

We return in this way to a situation similar to that we have started with: object c_i in Π_i and $X_k(i)$ in Π_{n+1}.

(2) If the rule i from m_j is of the form $A_i \to B_i$, and we use a rule $(a_0, out; b, in)$, for some $b \in T$, then the computation will never stop: we do not have a rule for introducing $X_j(i)$ alone in membrane $(i,1)$, hence $X_j(i)$ must release the trap object from membrane $(i,2)$. Therefore, we have to use the rule $(X'_j(i)a, out; X_j(i)a, in)$ from $R_{(i,1)}$ (at the same time, the object c_i comes to Π_i). As above, the three objects $X_j(i)c_ia$ can move together to Π_{n+1}, where, while c_i moves to Π_i, we exchange $X_j(1)\dots X_j(n)$ for some $X_k(1)\dots X_k(n)$ in Π_{n+1}, for $m_j \rightsquigarrow m_k$.

Also in this case we return to a situation similar to that we have started with: object c_i in Π_i and $X_k(i)$ in Π_{n+1}.

The process can be continued. Checking the correctness of the simulation of the rules in G is done in components Π_i, the fact that the rules which are simultaneously checked form a matrix of G is ensured by the component Π_{n+1}.

When a terminal matrix is simulated, component Π_{n+1} halts the computation by using the rule $(X_j(1)c_1a_1\dots X_j(n)c_na_n, in)$ (if we do not "hide" also the objects c_i in membrane $(n+1,1)$, then these objects have to go to components Π_i, where no rule can use them other than the trap-releasing ones).

We conclude that $L(G) = L(\Delta)$. □

5 Final Remarks

Let us first synthesize all previous results and remarks in a diagram – see Figure 4. The arrows indicate inclusions; if the arrow is marked with a dot, then that inclusion is known to be proper. The inclusions $RSM_n \subset RSM_{n+1}, n \geq 1$, are known to be proper, hence also the hierarchy $LdP_n^b, n \geq 1$, is infinite, but we do not know languages proving the strictness of inclusions $LdP_n^b \subseteq RSM_n \subseteq LdP_{n+1}^b, n \geq 1$, with the exception of the inclusion $RSM_1 \subset LdP_2^b$, because $RSM_1 = REG$ and LdP_2^b contains non-regular languages (see, e.g., the example in Section 3). Similarly, we do not know whether the inclusions $LdP_n \subseteq LdP_{n+1}, n \geq 2$, are proper – but we *conjecture* that this is the case.

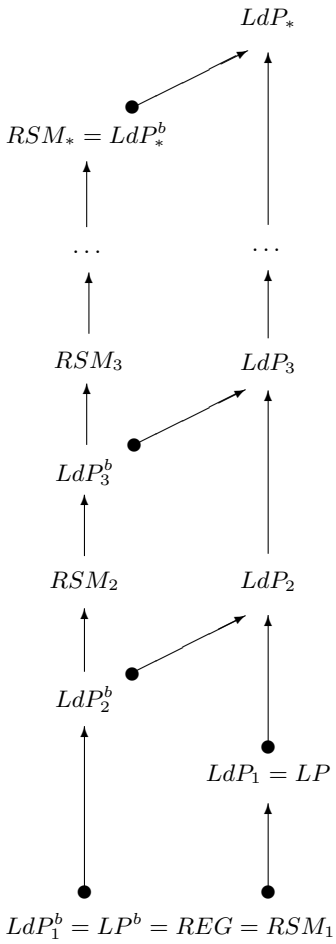


Fig. 4. The hierarchy of the families RSM_n, LdP_n^b, and LdP_n

Further open problems and research topics about dP systems can be found in the papers mentioned in the bibliography – the study of dP automata is one of the recently introduced and most active branches of membrane computing.

Acknowledgements

Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. E. Csuhaj-Varju, G. Vaszil: About dP automata
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, 1989.
3. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest University. Mathematics-Informatics Series*, 2010 (in press).
4. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing*. Springer, Berlin, 1997.
5. O. Ibarra: Simple matrix grammars. *Information and Control*, 17 (1970), 359–394.
6. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
7. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.
8. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Rainbow of Computer Science* (C.S. Calude, G. Rozenberg, A. Salomaa, eds.), LNCS, Springer, Berlin, 2010 (in press).
9. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP Systems. *Theoretical Computer Sci.*, in press.
10. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
11. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
12. The P Systems Website: `http://ppage.psystems.eu`.

Towards Bridging Two Cell-Inspired Models: P Systems and R Systems

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

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es

Summary. We examine, from the point of view of membrane computing, the two basic assumptions of reaction systems, the "threshold" and "no permanence" ones. In certain circumstances (e.g., defining the successful computations by local halting), the second assumption can be incorporated in a transition P system or in a symport/antiport P system without losing the universality. The case of the first postulate remains open: the reaction systems deal, deterministically, with finite sets of symbols, which is not of much interest for computing; three ways to introduce nondeterminism are suggested and left as research topics.

1 Introduction

The aim of this note is to bridge two branches of natural computing inspired from the biochemistry of a living cell, *membrane computing* (see, e.g., [11], [12], [13], and the domain website from [15]) and the recently introduced *reaction systems* area – see [2], [3], [4], [5], [6].

Both areas deals with populations of reactants (molecules) which evolve by means of reactions, with several basic differences. Most of these differences are not mentioned here (e.g., the compartmental structure of models – P systems – in membrane computing versus the missing of membranes in reaction systems – we also call them *R systems* –, the focus on evolution, not on computation, in reaction systems, the unique form of rules in reaction systems and so on), and we recall the two basic ones in the formulation from [2]:

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.

Passing from multisets, which are basic in P systems, to sets (actually, to multisets with an infinite multiplicity of their elements) 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. Moreover, the behavior of a reaction system is deterministic, from a set of symbols we precisely pass to a unique set of symbols (hence the behavior of a reaction system can be described by a graph of outdegree one, having the nodes marked with subsets of the total set of molecules). How to bridge at this level the two research areas (defining computations in reaction systems or working with multisets with infinite multiplicity of each element in P systems) remains as a research topic. Here we only propose three ways to introduce nondeterminism in reaction systems, so that more interesting computation (evolution) graphs can be obtained: providing tables of rules, considering also molecules with a finite multiplicity, and considering a threshold on the number of rules which can use simultaneously molecules of a given type.

P systems with sets were also considered in [10], mainly from the semantics (via Petri nets) point of view.

The second assumption of the reaction systems theory is much easier to handle in terms of membrane computing. The immediate idea is to simply remove 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 membrane computing. 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 consider here three possibilities: (i) the *local halting* of [8] (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 signals were considered in various papers; we refer here only to [9].

All these possibilities are checked both for transition and for symport/antiport P systems – with some cases still remaining open (the most important one is that of catalytic P systems).

2 Basic Definitions

For the sake of completeness, we recall here a few elementary notions about reaction systems and P systems.

The language theory notations are standard. An alphabet is a finite and nonempty set. For an alphabet V, by V^* we denote the set of all strings over V, including the empty string, denoted by λ. The set of nonempty strings over V is denoted by V^+. The length of a string $x \in V^*$ is denoted by $|x|$. The multisets over a finite set S are represented by strings in S^*; a string and all its permutations represent the same multiset. (The Parikh mapping of a string representing a multiset indicates the multiplicity of each object in the multiset.)

In the proofs from Section 4 we will use the characterization of recursively enumerable sets of numbers (sets of numbers computable by Turing machines; their family is denoted by NRE, reminding the fact that these sets ate length sets of recursively enumerable languages) by means of *register machines*; such a device is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction `HALT`), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- l_i : (`ADD`$(r), l_j, l_k$) (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- l_i : (`SUB`$(r), l_j, l_k$) (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- l_h : `HALT` (the halt instruction).

A register machine M computes (generates) a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by labels (and made possible by the content of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, i.e., they characterize the family NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its content.

2.1 Reaction Systems

We recall here some elementary notions and notation about reaction systems, as available in the few papers already published in this area – see again the titles mentioned at the beginning of the Introduction.

Let S be an alphabet (its elements are called *molecules* or, simply, symbols). A *reaction* (in S) is a triple $a = (R, I, P)$, where R, I, P are nonempty subsets of S such that $R \cap I = \emptyset$. R is the *reactant set of* a, I is the *inhibitor set of* a, and P is the *product set of* a. R, I, P are also denoted R_a, I_a, P_a. We denote by $rac(S)$ the set of all reactions in S.

If $T \subseteq S$ and $a \in rac(S)$, then a *is enabled by* T if $R_a \subseteq T$ and $I_a \cap T = \emptyset$, and then the *result of* a *on* T, denoted by $res_a(T)$, is defined by $res_a(T) = P_a$. If a is not enabled by T, then $res_a(T) = \emptyset$.

If A is a finite set of reactions, then *the result of* A *on* T is defined by $res_A(T) = \bigcup_{a \in A} res_a(T)$.

Then, a *reaction system* (we also call it an R *system*) is an ordered pair $\sigma = (S, A)$, where S is an alphabet and $A \subseteq rac(S)$.

Note in the definition of the result of a set A of reactions on a set T of molecules the occurrence of the two assumptions mentioned in the Introduction: a molecule can evolve by means of several reactions (or can inhibit several reactions if it appears in inhibitor sets), hence the multiplicity of each molecule is unbounded, while all molecules present at a given time "disappears", after the reactions we continue with the set of molecules produced by the reactions.

2.2 P Systems

We introduce first the class of transition P systems, closer in their definition to reaction systems. Some familiarity of the reader with the elementary notions of membrane computing is assumed, e.g., from [12], [13].

A membrane structure is a cell-like hierarchical arrangement of labeled membranes (understood as 3D vesicles); the external membrane is usually called the *skin* membrane, and a membrane without any membrane inside is called *elementary*. With each membrane, a *region* is associated, the space delimited by it and the inner membranes, if any. A membrane structure can be represented by a rooted tree or by an expression of labeled parentheses (with a unique external parenthesis, associated with the skin).

Given an alphabet O of *objects*, a multiset-rewriting rule (over O; we also say *evolution rule*) is a pair (u, v), written in the form $u \to v$, where u and v are multisets over O (given as strings in O^*). The rules are classified according to the complexity (of their left hand side). A rule with at least two objects in its left hand side is said to be *cooperative*; a particular case is that of *catalytic* rules, of the form $ca \to cv$, where c is a catalyst which assists the object a (which is not a catalyst) to evolve into the multiset v (where no catalyst appears); rules of the form $a \to v$, where a is an object, are called *non-cooperative*.

The rules can also have associated promoters or inhibitors, objects whose presence make possible the use of a rule (but are not modified by the rule application), respectively, can forbid the application of the rule. Also, a *priority* relation can be considered, in the form of a partial order relation among the set of rules in a membrane; a rule can be used only if no rule of a higher priority can be used. Finally, we mention the *dissolution* operation: a rule can be of the form $u \to v\delta$ and, when used, the membrane in which it is applied is "dissolved", its objects become elements of the immediately higher membrane (and its rules disappear, as being associated with the "reactor" defined by the membrane). We do not enter here into details – in general, such additional controls on using the rules are rather useful (and powerful) in "programming" the work of a P system.

Now, a *transition P system* (of degree m) is a construct

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

where O is the alphabet of objects, μ is the membrane structure (with m membranes), given as an expression of labeled parentheses, $w_1, \ldots, w_m$ are (strings over O representing) multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of evolution rules associated with the regions of μ, and i_{in}, i_{out} are the labels of input and output membranes, respectively. If the system is used in the generative mode, then i_{in} is omitted, and if the system is used in the accepting mode, then i_{out} is omitted. If the system is a catalytic one, then a subset C of O is specified, containing the catalysts. The number m of membranes in μ is called the *degree* of Π.

The rules in sets R_i are of the form $u \to v$, as specified above, with $u \in O^+$, but with the objects in v also having associated *target indications*, i.e., $v \in (O \times \{here, out, in\})^*$. After using a rule $u \to v$, the objects in u are consumed, and those in v are produced; if $(a, here)$ appears in v, then a remains in the same compartment of the system where the rule was used, if (u, out) is in v, then the object a is moved immediately in the region surrounding the compartment where the rule was used (this is the environment if the rules is used in the skin region), and if (a, in) is in v, then a is sent to one of the inner membranes, nondeterministically chosen (if there is no membrane inside the membrane where the rule is meant to be applied, then the use of the rule is forbidden). The indication *here* is omitted, we write a instead of $(a, here)$.

The rules are used in the nondeterministic maximally parallel manner: in each membrane, a multiset of rules is applied such that there is no larger multiset of rules which is applicable in that membrane.

In the generative mode, the result of a computation consists of the number of objects in membrane i_{out} in the moment when the computation halts, i.e., no rule can be applied in any membrane of the system. In the accepting mode, a number is introduced in the membrane i_{in}, in the form of the multiplicity of a given object, and, if the computation halts, then this number is accepted. A P system can also be used in the computing mode, with a number introduced in membrane i_{in} and the result obtained in membrane i_{out}, in the moment when the computation halts.

In what follows, we only deal with generating P systems. One knows that catalytic P systems are Turing equivalent, they compute all recursively enumerable sets of natural numbers (i.e., they characterize NRE), but non-cooperative P systems compute only semilinear sets of numbers. Details can be found in the references given at the beginning of the Introduction.

Another much investigated class of P systems is that of *symport/antiport P systems*. These systems are not based on reaction rules, but on biological operations of passing coupled molecules across membranes.

We can formalize these operations by considering *symport* rules of the form (x, in) and (x, out), and *antiport* rules of the form $(z, out; w, in)$, where x, z, and w are multisets of objects.

A *P system with symport/antiport rules* is a construct of the form

$$\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_{in}, i_{out}),$$

where all components $O, \mu, w_1, \ldots, w_m, i_{in}, i_{out}$ are as in a P system with multiset rewriting rules, $E \subseteq O$, and $R_1, \ldots, R_m$ are finite sets of symport/antiport rules associated with the m membranes of μ. The objects of E are supposed to be present in the environment of the system with an arbitrary multiplicity. (Note that the symport/antiport rules do not change the number of objects, but only their place, that is why we need a supply of objects in the environment; this supply is inexhaustible, i.e., does not matter how many objects are introduced in the system, arbitrarily many still remain in the environment.)

As above, the rules are used in the nondeterministic maximally parallel manner: we choose nondeterministically multisets of rules associated with each membrane and such an m-tuple of multisets is applied if for no membrane a rule can be added to the associated multiset still having the enlarged m-tuple of multisets applicable. We define transitions, computations, and halting computations in the usual way. The number of objects present in region i_{out} in the halting configuration is said to be computed by the system by means of that computation; the set of all numbers computed in this way by Π is denoted by $N(\Pi)$. Accepting and computing symport/antiport P systems are defined in the natural manner.

It is known that symport/antiport P systems (with a small number of membranes and with rules of a low complexity) characterize NRE.

Note that in the previous definitions multisets play a crucial role, objects not evolving by a rule remain unchanged, and that always successful computations are defined by halting.

3 Computing with Reaction Systems

Starting from a reaction system $\sigma = (S, A)$, we can consider a "generative device" $\gamma = (S, A, w_0)$, where w_0 is a subset of S, an "axiom set". (We denoted the starting set by a small letter, like a string, in the multiset sense, because we will need such

an approach below, e.g., when part of molecules will be considered in the multiset sense.) Then, we can obtain a sequence $w_0 \Longrightarrow_A w_1 \Longrightarrow_A w_1 \Longrightarrow_A \ldots$, where $w_{i+1} = res_A(w_i), i \geq 0$.

Two basic observations: (i) this sequence is unique, because the passage from a set of molecules to the next one is deterministic, and (ii) for all $i \geq 0$ we have $w_i \subseteq S$. Therefore, if we associate a label to each subset of S, then a sequences as above is either finite (at some moment, no rule can be applied, all elements vanishes, hence we end with the label of the empty set), or the sequence is infinite and then it can be described by a string of the form uv^ω: after a finite path among subsets of S, we enter a cycle which goes forever.

In terms of graphs, the relation $\Longrightarrow_A$ defines a graph $G_S(A) = (2^S, \Longrightarrow_A)$ of outdegree (at most) one (the outdegree can be zero, but this is a trivial case). Computations in $\gamma = (S, A, w_0)$ can then be followed along the paths in $G_S(A)$ starting in the node w_0.

We do not have here too much from a computability point of view, even if we consider the graph itself as the result of the computation (the number of graphs $G_S(A)$ is bounded, because of the finiteness of S). The dramatic restriction here is the deterministic behavior of a reaction system, that is why we propose here three possibilities to get a nondeterministic device.

The first natural idea is to consider a *tabled* reaction system, in the form $\gamma = (S, A_1, A_2, \ldots, A_n, w_0)$ where $A_i, 1 \leq i \leq n$, are sets of reactions over S (called tables). Like in an E0L system (see, e.g., [14]), in a step of a computation we can nondeterministically choose the table to use, hence branching is possible. In this case, we can also introduce halting as a criterion for defining successful computations: a halt table can be considered, for instance, with rules of the form $a \to a'$ for all $a \in S$, such that no rule exists for a', hence in the next step all (primed) molecules disappear.

Another idea, at the bridge of membrane computing and reaction systems, is to consider a subset $C \subseteq S$ of molecules for which the multiplicity matters, and having finite multiplicities. Then we move towards usual P systems (cooperative, with inhibitors, hence rather powerful). The elements of C are counted when applying the rules, those in $S - C$ not. The nondeterminism appears now when using copies of elements in C, if more rules than such objects can be applied.

Finally, without modifying the components of a computing reaction system $\gamma = (S, A, w_0)$, we can provide the nondeterminism by introducing a general threshold on the number of rules which can use the same molecule, hence having a system of the form $\gamma = (S, A, w_0, k)$, where k is the threshold. This is similar to the previous case, taking $C = S$, which is like working with multisets, but with the same multiplicity for all objects (only at most k copies of each object can evolve, the others are removed, hence we can assume that the multiplicity is exactly k for each object). The nondeterminism appears again when choosing the rules which compete for the same objects. We have a usual P system, but dealing with finite populations of objects: if only k rules are used for each molecule, only finitely many

rules are used, all existing objects are consumed or they vanishes and a bounded number of objects are produced.

In the second case, the multiplicity of objects in C can increase arbitrarily, but in the other two cases we again deal with a finite computation graph (but not of an outdegree bounded in advance).

All these three possibilities remain to be investigated: properties of the obtained graphs, possible links with computing devices from formal language and automata theory, influence of the introduced parameters (number of tables, cardinality of C, threshold k), possible hierarchies.

Of course, another research topic is to find other ways of building a (string or graph) computing device in terms of reaction systems.

4 P Systems without the "Permanence" of Objects

Let us now move to membrane computing, and borrow from reaction systems area the assumption that an object which is not involved in a rule does not pass to the next configuration. Then, we cannot define the result of a computation by halting, because in a halting step all objects vanish. Similarly, it is not enough to add dummy rules of the form $a \to a$ (in transition systems), because this time the computation never halts. Thus, we have to define successful computations by other conditions – and we consider here the three possibilities recalled in the Introduction: local halting, signal-objects, signal-events. The definitions are straightforward, we pass directly to examine the power of P systems endowed with such conditions.

4.1 The Case of Transition P Systems

Let us consider a register machine $M = (m, H, l_0, l_h, I)$, as introduced at the beginning of Section 2. We first construct a transition P system $\Pi = (O, \mu, w_1, w_2, R_1, R_2, 1)$, aiming to simulate the machine M, and then we discuss modes of defining the result of a computation in Π. We take:

$$
\begin{aligned}
O &= \{a_i, a'_i \mid 1 \leq i \leq m\} \cup \{l, l', l'', l''', l^{iv} \mid l \in H\} \cup \{b, c, \#\},\\
\mu &= [\,[\quad]_2\,]_1,\\
w_1 &= l_0,\ w_2 = b,\\
R_1 &= \{l_i \to l_j a_r,\\
&\qquad l_i \to l_k a_r \mid l_i : (\texttt{ADD}(r), l_j, l_k) \in I\}\\
&\quad \cup \{a_1 \to a_1\} \cup \{a_s \to a_s a'_s \mid 2 \leq s \leq m\}\\
&\quad \cup \{l_i \to l'_i l''_i,\\
&\qquad l'_i a_r \to l'''_i,\\
&\qquad l'_i a'_r \to (\#, in),
\end{aligned}
$$

$$
\begin{aligned}
& l_i'' \to l_i^{iv}, \\
& l_i^{iv} \to l_k, \\
& l_i''' \to (\#, in), \\
& l_i^{iv} l_i''' \to l_j \mid l_i : (\mathtt{SUB}(r), l_j, l_k) \in I\} \\
\cup\ & \{l_h \to (l_h, in)\}, \\
R_2 = \{& b \to b,\ \# \to \#,\ l_h b \to c\}.
\end{aligned}
$$

This system works as follows. The contents of each register r is represented by the number of occurrences of objects a_r in the skin region of Π. In each step, each of these objects is reproduced, hence their number is never decreased; moreover, objects $a_r, r \neq 1$, also produce "twin objects" a_r', which disappear in the next step (one copy is used in simulating SUB instructions, as we will see below). Object b evolves forever in membrane 2. One of our goals is to define the end of a computation in Π by local halting, namely, by halting the evolution of membrane 2. This can happen only in the presence of the halt label of M, and without introducing the trap-object #.

We start with label l_0 in membrane 1. In general, when a label l_i is present in membrane 1, the respective instruction of M is simulated.

The simulation of an ADD instruction is obvious. Assume that l_i is the label of a SUB instruction, $l_i : (\mathtt{SUB}(r), l_j, l_k)$. We use the rule $l_i \to l_i' l_i''$. At the same time, all objects a_s' disappear and all objects a_s are replaced by $a_s a_s'$, $2 \leq s \leq m$; objects a_1 remains always unchanged during simulating a SUB instruction (remember that M never decreases register 1). In the next step, l_i'' is replaced by l_i^{iv}, while l_i' has two possibilities. If a copy of a_r is present (hence register r is not empty), then also a_r' is present. If the rule $l_i' a_r \to l_i'''$ is used, then a_r' disappear, and this is the correct continuation – in the next step, the rule $l_i^{iv} l_i''' \to l_j$ is used, introducing the label of the next instruction to simulate. If, instead of $l_i' a_r \to l_i'''$, the rule $l_i' a_r' \to (\#, in)$ is used, then the trap-object # is introduced in membrane 2, and it will evolve here forever. If the register r is empty, hence no object a_r and a_r' is present, then l_i''' is not introduced, l_i' disappears. In the next step l_i^{iv} has to evolve by means of the rule $l_i^{iv} \to l_k$, the correct continuation in the register machine. If this rule is used also in the presence of l_3''' (hence in case the register r was nonempty), then we have to use the rule $l_i''' \to (\#, in)$.

In this way, the instructions of M are correctly simulated. When the halt label l_h is introduced in M, this object is moved to membrane 2. If the only object present here is b, then the computation in membrane 2 can halt by means of $l_h b \to c$. If also # is present, then the computation in membrane 2 continues forever. The number of objects a_1 in membrane 1 at the moment of halting membrane 2 gives the result of the computation. (Remember that all registers of M except the first one are empty in the end of computations in M.)

The previous construction can be slightly modified in order to mark the end of the computation by means of signal objects or events instead of local halting. For instance, if we replace the rule $\# \to \#$ of R_2 with $\# \to \delta$, then membrane

2 is dissolved, the rule $l_h b \to c$ cannot be used. Thus, the signal can be either the object c or the use of the rule $l_h b \to c$. When one of these signals appears in membrane 2, the number of copies of a_1 in membrane 1 is the result of the computation. If # was introduced, then these signals never appear.

We conclude with the assertion-theorem that *transition P systems of degree 2, using cooperative rules, without the "permanence" of objects, are computationally complete.*

An interesting *open problem* in this framework is the case of catalytic P systems, known to be universal in the "permanence" assumption (see, e.g., [7]).

4.2 The Case of Symport/Antiport P Systems

The case of symport/antiport systems just "recodes" the previous construction, but, because for these systems we do not have the dissolution operation (it can be introduced, in a natural way, but this was not done up to now, hence we do not consider it here), only the case of local halting is considered.

Take again a register machine $M = (m, H, l_0, l_h, I)$. We construct the symport/antiport P system $\Pi = (O, \mu, w_1, w_2, E, R_1, R_2, 1)$ with the same alphabet of objects and membrane structure as in the previous subsection, but with $w_1 = bl_0,\ w_2 = b,\ E = O$, and with the rules as specified in Figure 1 – instead of a formal definition, we give now the graphical representation of the system.

The functioning of this system is very much similar to the functioning of the system in the previous subsection, hence we do not describe it in details (membrane 2 halts only when # is not present and l_h moves outside the system the object b from the skin region).

The case of defining the result of a computation by means of signals – objects or events (using a specified rule) – remains as an *open problem.* (Considering a priority relation on each set of rules can easily solve this problem.) The previous symport/antiport P system contains antiport rules of sizes (2, 1) and (1, 2), which is "large" for universality results in the case when objects are persistent (see, e.g., [1]). Can the size of rules be decreased also in the case discussed here?

5 Final Remarks

Although there are so many similarities and differences between membrane computing (P systems) and reaction systems (R systems), up to our knowledge, so far there is no bridging investigation, in spite of the fact that this research topic was formulated several times in the membrane computing community (e.g., during the yearly Brainstorming Weeks on Membrane Computing). This is a natural and surely fruitful area to explore, especially in checking the influence of basic postulates of one domain in another one and in borrowing notions and research issues from a domain to another one. The present paper is only a first step in this direction, examining the two basic postulates of reaction systems: working with

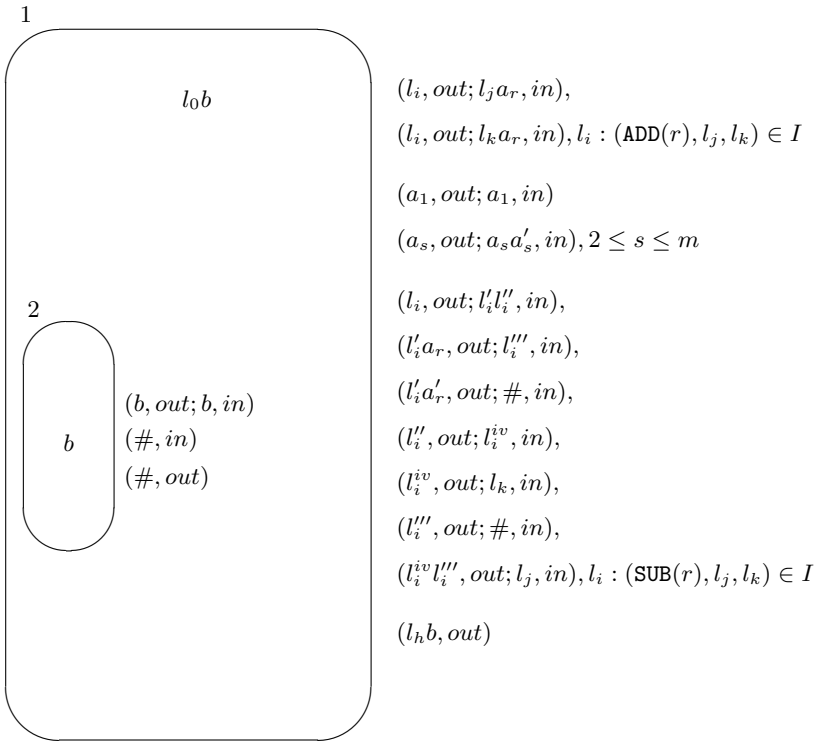


Fig. 1.

molecules whose multiplicity is not counted (it is considered infinite) and removing from the system molecules which do not evolve by reactions. Many open problems and research topics are formulated.

Acknowledgements. Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin: Some optimal results on symport/ antiport P systems with minimal cooperation. In *Cellular Computing (Complexity Aspects)* (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, eds.), ESF PESC Exploratory Workshop, Fénix Editorial, Sevilla, 2005, 23–36.
2. A. Ehrenfeucht, G. Rozenberg: Basic notions of reaction systems, *Proc. DLT 2004* (C.S. Calude, E. Calude, M.J. Dinneen, eds.), LNCS 3340, Springer, 2004, 27–29.
3. A. Ehrenfeucht, G. Rozenberg: Reaction systems. *Fundamenta Informaticae*, 75 (2007), 263–280.

4. A. Ehrenfeucht, G. Rozenberg: Events and modules in reaction systems. *Theoretical Computer Sci.*, 376 (2007), 3–16.
5. A. Ehrenfeucht, G. Rozenberg: Introducing time in reaction systems. *Theoretical Computer Sci.*, 410 (2009), 310–322.
6. A. Ehrenfeucht, G. Rozenberg: Reaction systems. A model of computation inspired by biochemistry. *Proc. DLT 2010* (Y. Gao et al., eds.), LNCS 6224, Springer, 2010, 1–3.
7. R. Freund, L. Kari, P. Sosik: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Sci.*, 330 (2005), 251–266.
8. R. Freund, M. Oswald: Partial halting in P systems. *Intern. J. Foundations of Computer Sci.*, 18 (2007), 1215–1225.
9. P. Frisco: *Computing with Cells. Advances in Membrane Computing*. Oxford University Press, 2008.
10. J. Kleijn, M. Koutny: Membrane systems with qualitative evolution rules, *Fundaenta Informaticae*, to appear.
11. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (first circulated as Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
12. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
13. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
14. G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
15. The P Systems Website: `http://ppage.psystems.eu`.

Smoothing Problem in 2D Images with Tissue-like P Systems and Parallel Implementation

Francisco Peña-Cantillana[1], Daniel Díaz-Pernil[2], Hepzibah A. Christinal[2,3], Miguel A. Gutiérrez-Naranjo[1]

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

[2] Computational Algebraic Topology and Applied Mathematics Research Group
Department of Applied Mathematics I
University of Sevilla, Spain
sbdani@us.es

[3] Karunya University, Coimbatore, Tamilnadu, India
hepzi@yahoo.com

Summary. Smoothing is often used in Digital Imagery to reduce noise within an image. In this paper we present a Membrane Computing algorithm for smoothing 2D images in the framework of tissue-like P systems. The algorithm has been implemented by using a novel device architecture called CUDA™, (Compute Unified Device Architecture). We present some examples, compare the obtained time and present some research lines for the future.

1 Introduction

The study of digital images [22] has seen a large progress over the last decades. The aim of dealing with an image in its digital form is improving its quality, in some sense, or to simply achieving some artistic effect. The physical properties of camera technology are inherently linked to different sources of noise, so the application of smoothing algorithm are necessary for an appropriate use of the images. Smoothing is often used to reduce such noise within an image.

In this paper we use Membrane Computing[4] techniques for smoothing 2D images in the framework of tissue-like P systems. The algorithm has been implemented by using a novel device architecture called CUDA™, (Compute Unified Device Architecture) [16, 23]. CUDA™ is a general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processors Units (GPUs)

[4] We refer to [19] for basic information in this area, to [20] for a comprehensive presentation and the web site [24] for the up-to-date information.

to solve many complex computational problems[5] in a more efficient way than on a CPU . This parallel architecture has been previously used in Membrane Computing [1, 2, 3] but, to the best of our knowledge, this is the first time that it is used for implementing smoothing algorithms with tissue-like P systems.

Dealing with Digital Imagery has several features which make it suitable for techniques inspired by nature. One of them is that it can be parallelized and locally solved. Regardless how large the picture is, the segmentation process can be performed in parallel in different local areas. Another interesting feature is that the basic necessary information can be easily encoded by bio-inspired representations.

In the literature, one can find several attempts for bridging problems from Digital Imagery with Natural Computing as the works by K.G. Subramanian *et al.* [4, 5] or the work by Chao and Nakayama where Natural Computing and Algebraic Topology are linked by using Neural Networks [6] (extended Kohonen mapping). Recently, new approaches have been presented in the framework of Membrane Computing [7, 11]. In [8, 9, 10], Christinal *et al.* started a new bio-inspired research line where the power and efficiency of tissue-like P systems [12, 13] were applied to topological processes for 2D and 3D digital images.

The paper is organised as follows: Firstly, we recall some basics of tissue-like P systems and the foundations of Digital Imagery. Next we present our P systems family and an easy example showing different results by using different thresholds. In Section 3 we present the implementation in CUDA™ of the algorithm and show an illustrative example, including a comparative of the time obtained in the different variants. The paper finishes with some final remarks and hints for future work.

2 Preliminaries

In this section we provide some basics on the used P system model, tissue-like P systems, and on the foundation of Digital Imagery.

Tissue-like P systems [14, 15] have two biological inspirations: intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a network of processors dealing with symbols and communicating these symbols along channels specified in advance.

Formally, a *tissue-like P system* with input of degree $q \geq 1$ is a tuple

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

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*;
2. $\Sigma(\subset \Gamma)$ is the input alphabet;
3. $\mathcal{E} \subseteq \Gamma$ is the alphabet of objects in the environment;
4. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration;

[5] For a good overview, the reader can refer to [17, 18].

5. $\mathcal{R}$ is a finite set of communication rules of the following form:

$$(i, u/v, j)$$

for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;
6. $i_\Pi \in \{1, 2, \ldots, q\}$ is the input cell;
7. $o_\Pi \in \{0, 1, 2, \ldots, q\}$ is the output cells

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

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

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

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

A *configuration* is an instantaneous description of the P system Π. Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A sequence of computation steps is called a *computation*. A configuration is *halting* when no rules can be applied to it. Then, a computation halts when the P system reaches a halting configuration.

Next we recall some basics on Digital Imagery[6].

A *point set* is simply a topological space consisting of a collection of objects called points and a topology which provides for such notions as *nearness* of two points, the *connectivity* of a subset of the point set, the *neighborhood* of a point, *boundary points*, and *curves* and *arcs*. For a point set X in Z, a *neighborhood function* from X in Z, is a function $N : X \to 2^Z$. For each point $x \in X$, $N(x) \subseteq Z$. The set $N(x)$ is called a *neighborhood* for x.

There are two neighborhood function on subsets of $\mathbb{Z}^2$ which are of particular importance in image processing, the *von Neumann* neighborhood and the *Moore* neighborhood. The first one $N : X \to 2^{\mathbb{Z}^2}$ is defined by $N(x) = \{y : y = (x_1 \pm j, x_2)$ *or* $y = (x_1, x_2 \pm k),\ j, k \in \{0, 1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbb{Z}^2$. While the Moore neighborhood $M : X \to 2^{\mathbb{Z}^2}$ is defined by $M(x) = \{y : y = (x_1 \pm j, x_2 \pm k),\ j, k \in \{0, 1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbb{Z}^2$. The von Neumann

[6] We refer the interested reader to [21] for a detailed introduction.

and Moore neighborhood are also called the *four neighborhood* (4-adjacency) and *eight neighborhood* (8-adjacency), respectively.

An *Z-valued image* on X is any element of Z^X. Given an Z-valued image $I \in Z^X$, i.e. $I : X \to Z$, then Z is called the set of possible range values of I and X the spatial domain of I. The graph of an image is also referred to as the *data structure representation* of the image. Given the data structure representation $I = \{(x, I(x)) : x \in X\}$, then an element $(x, I(x))$ is called a *picture element* or *pixel*. The first coordinate x of a pixel is called the *pixel location* or *image point*, and the second coordinate $I(x)$ is called the *pixel value* of I at location x.

For example, X could be a subset of $\mathbb{Z}^2$ where $x = (i, j)$ denotes spatial location, and Z could be a subset of $\mathbb{N}$ or $\mathbb{N}^3$, etc. So, given an image $I \in Z^{\mathbb{Z}^2}$, a pixel of I is the form $((i, j), I(x))$, which be denoted by $I(x)_{ij}$. We call the *set of colors* or *alphabet of colors of* I, $\mathcal{C}_I \subseteq Z$, to the image set of the function I with domain X and the image point of each pixel is called *associated color*. We can consider an order in this set. Usually, we consider in digital image a predefined alphabet of colors $\mathcal{C} \subseteq Z$. We define $h = |\mathcal{C}|$ as the size (number of colors) of $\mathcal{C}$. In this paper, we work with images in grey scale, then $\mathcal{C} = \{0, \ldots, 255\}$, where 0 codify the black color and 255 the white color.

A *region* could be defined by a subset of the domain of I whose points are all mapped to the same (or similar) pixel value by I. So, we can consider the region R_i as the set $\{x \in X : I(x) = i\}$ but we prefer to consider a region r as a maximal connected subset of a set like R_i. We say two regions r_1, r_2 are adjacent when at less a pair of pixel $x_1 \in r_1$ and $x_2 \in r_2$ are adjacent. We say x_1 and x_2 are *border pixels*. If $I(x_1) < I(x_2)$ we say x_1 is an *edge pixel*. The set of connected edge pixels with the same pixel value is called a *boundary* between two regions.

The purpose of image enhancement is to improve the visual appearance of an image, or to transform an image into a form that is better suited for human interpretation or machine analysis. There exists a multitude of image enhancement techniques, as are averaging of multiple images, local averaging, Gaussian smoothing, max-min sharpening transform, etc.

One of the form to enhancement an image could be eliminate non important regions of an image, i.e., remove regions which do not provide relevant information. This technique is known as smoothing.

2.1 A Family of Tissue-like P Systems

Given a digital image with n^2 pixels and $n \in \mathbb{N}$ we define a tissue-like P system whose input is given by the pixels of the image encoded by the objects a_{ij}, where $1 \leq i, j \leq n$ and $a \in \mathcal{C}$. Next, we shall give some outlines how to prove that our *smoothing problem* can be solved in a logarithmic number of steps using a family of tissue-like P systems Π.

We define a family of tissue-like P systems to do an smoothing of a 2D image. For each image of size n^2 with $n \in \mathbb{N}$, we consider the tissue-like P system with input of degree 1:

$$\mathbf{\Pi}(r,n) = (\Gamma, \Sigma, \mathcal{E}, w_1, \mathcal{R}, i_\Pi, o_\Pi),$$

where

- $\Gamma = \Sigma \cup \mathcal{E}$,
- $\Sigma = \{a_{ij} : a \in \mathcal{C},\ 1 \leq i,j \leq n\}$,
- $\mathcal{E} = \{a_{ij} : 1 \leq i,j \leq n,\ a \in \mathcal{C}\}$,
- $w_1 = \emptyset$,
- R is the following set of communication rules:
  - $(1, a_{ij}b_{kl}/a_{ij}a_{kl}, 0)$,
    - for $1 \leq i,j \leq n$,
    - $a,b \in \mathcal{C}$, $a < b$ and $d(a,b) \leq r$,

    These rules are used to simplify the image. If we have two colors whose distance in the alphabet of colors of the image is small, then we change the high color by the small one. Of this manner, we change the regions structure.
- $i_\Pi = o_\Pi = 1$.

Each P system works as follows: We take pairs of adjacent pixels and change the color of the pixel with lower color. We do it in a parallel way with all the possible pairs of pixels. In the next step, we will repeat the previous process, but the colors of the pixels have could to be changed. So, we need, in the worst case, a linear number of steps to do all the possible changes to obtain an smoothing of our image.

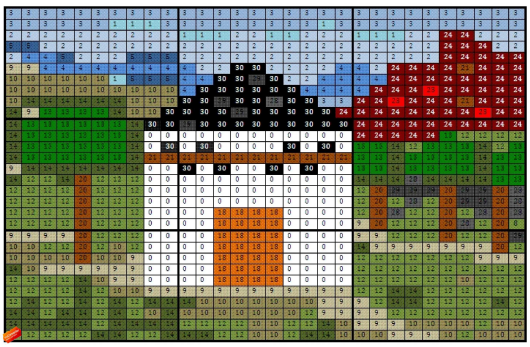


Fig. 1. An example

2.2 An Easy Example

In this section, we show the results obtained by the application of our method. Our input image (of size 30×30) can be seen in Figure 1. In this case, 0 represents

to white color and 30 represents to black color, i.e., the inverse order for $\mathcal{C}$ is considered.

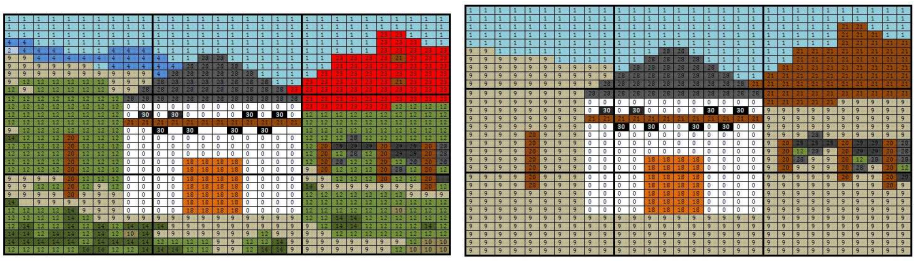


Fig. 2. Theoretical Smoothness of the Figure 1.

Working with different thresholds provides different results as we can observe in Figure 2. If we take $r = 5$, then we get the first image, and when the threshold is $r = 10$ the second image is obtained. By using this method, it is clear that the structure of regions is (more or less) conserved with a threshold of $r = 5$ and we need to a high number to obtain a more simplified image. Moreover, we can observe in both cases the color of regions is similar to the regions of input image in this method. Therefore, we can use this technique for smoothing images, and clarify the structure of a region without eliminate important information.

Bearing in mind the size of the input data is $O(n^2)$, $|\mathcal{C}| = h$ and r is the threshold used with both membrane solutions. The amount of necessary resources for defining the systems of our families and the complexity of our problem is determined in the following table:

Smoothing Problem	
Complexity	Dynamical
Number of steps of a computation	$O(n)$
Necessary Resources	
Size of the alphabet	$n^2 \cdot h$
Initial number of cells	1
Initial number of objects	0
Number of rules	$O(n^2 \cdot h)$
Upper bound for the length of the rules	4

3 Parallel Implementation

GPUs constitute nowadays a solid alternative for high performance computing, and the advent of CUDA™ allow programmers a friendly model to accelerate a broad range of applications. The way GPUs exploit parallelism differ 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. GPUs can support several thousand of concurrent threads providing a massively parallel environment. This parallel computation model leads us to look for a highly parallel computational technology where a parallel simulator can run efficiently.

In this paper, we present a parallel software tool based in our membrane solution for smoothing images. It has been developed by using Microsoft Visual Studio 2008 Professional Edition (C++) with the plugging Parallel Nsight (CUDA™) under Microsoft Windows 7 Professional with 32 bits.

To implement the P systems, CUDA™ C, an extension of C for implementations of executable kernels in parallel with graphical cards NVIDIA has been used. It has been necessary the *nvcc compiler* of CUDA™ Toolkit. Moreover, we use libraries from openCV to the treatment of input and output images. Microsoft Visual Studio 2008 is responsible for calling to the compilers to build the objects, and to link them with the final program. This allows us to deal with images stored in .BMP, .DIB, .JPEG, .JPG, .JPE, .PNG, .PBM, .PGM, .PPM, .SR, .RAS, .TIFF and .TIF formats

The experiments have been performed on a computer with a CPU Intel Pentium 4 650, with support for HT technology which allows to work like two CPUs of 32 bits to 3412 MHz. Computer has 2 MB of L2 cache memory and 1 GB DDR SDRAM of main memory with 64 bits bus wide to 200 MHz. Moreover, it has a hard disc of 160 GB SATA2 with a transfer rate of 300 Mbps in a 8 MB buffer.

The graphical card (GPU) is an NVIDIA Geforce 8600 GT composed by 4 *Stream Processors* with a total of 32 cores to 1300 MHz and executes 512 threads per block as maximum. It has a 512 MB DDR2 main memory, but 499 MB could be used by processing in a 128 bits bus to 700 MHz. So, the transfer rate obtained is by 22.4 Gbps. For constant memory used 64 KB and for shared memory 16 KB (It is not a good data for a good CUDA™ graphical card). Its Compute Capability is 1.1 (from 1.0 to 2.1), then we can obtain a lot of improvements in the efficiency of the algorithms.

If we compare CPU and GPU, we observe that the former has two cores to 3412 MHz and the latter has 32 to 1300Mhz and has larger memory.

We have developed two applications of our P systems. In this case, we consider the natural order in the set of colors $\mathcal{C} = \{0, \ldots, 255\}$. In the first one, we have considered a deterministic implementation, where we work with the Moore neighborhood. So, the system checks if the rules can be applied for eight adjacent pixels. In the second one, we have considered an random selection of an adjacent pixel to work. Then, the system checks only one possibility chosen in an random way. Of this form, we simulate the characteristic non determinism of P systems with random. Moreover, we have decided to stop the system before the halting configuration, because more than an appropriate number of parallel steps of processing could be non-operative. In fact, in the deterministic version, the process could fin-

ish before the pre-fixed number of steps. So, the system needs some time to check this possibility. In the second one, it is not necessary to look at this question.

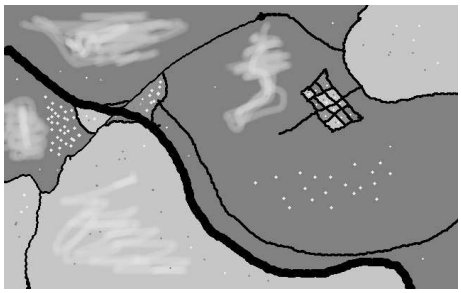


Fig. 3. Original image

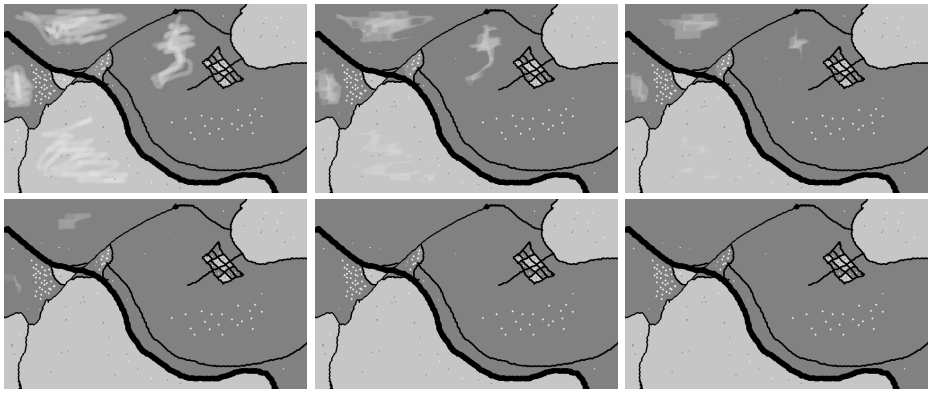


Fig. 4. Deterministic version, Threshold 50: 0, 5, 10, 20, 32 and 44 steps, respectively.

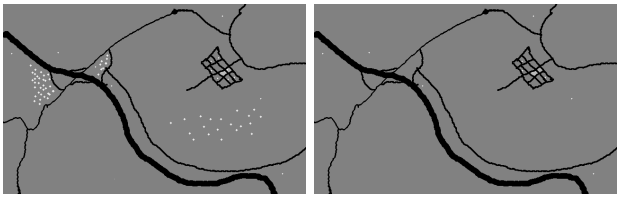


Fig. 5. Deterministic version. First: Threshold 75, step 193. Second: Threshold 125, step 193.

We consider the image of size 640×400 in Figure 3. When we take the deterministic application of our software, we can check that if we use an threshold $r = 50$, our software smooths the original image (see Figure 4) using 44 parallel steps. Nonetheless, when we work with a higher threshold, new important regions are changed, and the output image is different. (See the images of Figure 5).

When we consider the random version of our software, we can check that if we use an threshold $r = 50$ we need 300 steps, but the differences with the resulting image with 150 or 200 steps are minimum, as we can see in Figure 6. When we take higher thresholds, as in the Figure 7, we can check that new regions change of colors.

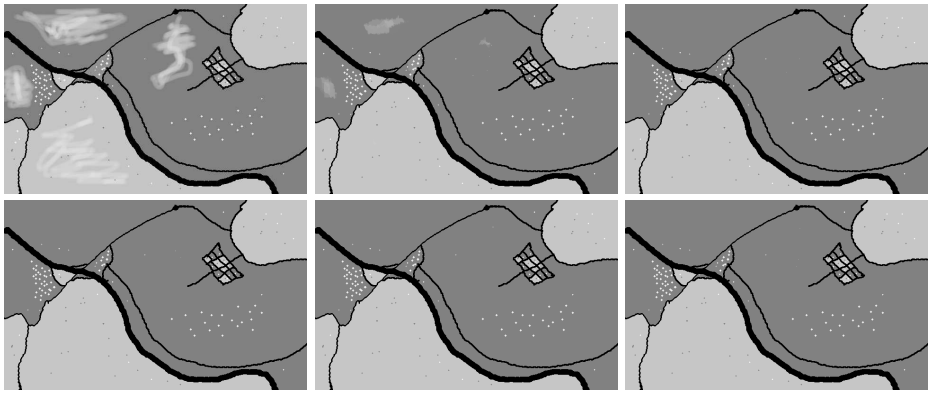


Fig. 6. Random version, Threshold 50: 0, 50, 100, 150, 200 and 300 steps, respectively.

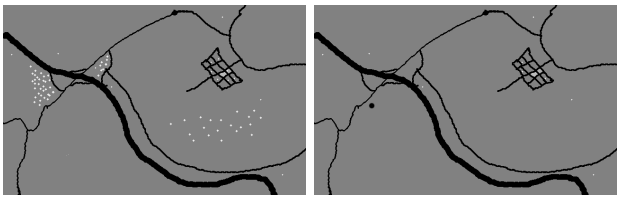


Fig. 7. Random version.First: Threshold 75, step 1000. Second: Threshold 125, step 800.

We present a study the running time of our software for both cases with different thresholds, showed in the above examples, with the next table. We can observe that deterministic version of our software needs less time with respect to the random version. In the first one, it applies eight rules for each pixels while, in the second one, it applies only one rule for each pixel. Moreover, we need a running time to implement the random in each step for each pixel.

Version \ Thresholds	Computation steps	Running Time
Determ. \ 50	44	536.522 ms
Determ. \ 75	193	1582.098 ms
Determ. \ 125	193	1563.660 ms
Random \ 50	300	6823.683 ms
Random \ 75	1000	21537.701 ms
Random \ 100	800	17332.891 ms

Finally, we have done some experiments with our software to know what happened if we work with images of different size. We have checked our software with images until size 512 in both version. The deterministic version needs much time with bigger images, and the random version does not work with those images. This is a physical problem our graphical card, because the shared memory is small.

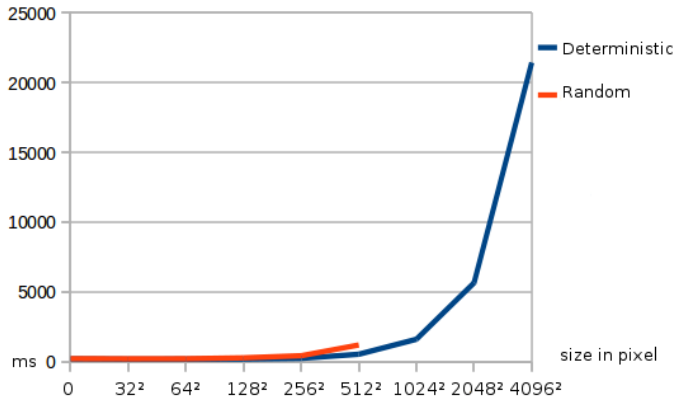


Fig. 8. Comparatives.

4 Conclusions and Future Work

In this paper, three emergent research fields are put together. Firstly, as pointed in [8, 10], Membrane Computing has 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 the intrinsic parallelism of Membrane Computing techniques cannot be implemented in current one-processor computers, so the potential advantages of the theoretical design are lost.

In this paper we show that the drawback of using one-processor computers for implementing Membrane Computing designs can be avoided by using the parallel architecture CUDA™. This new technology provides the hardware needed for a real parallel implementation of Membrane Computing algorithms.

Considering this paper as a starting point, several research lines are open: From Digital Imagery, new parallel algorithms can be proposed adapted to the new technology, from the Membrane Computing side, new design or different P system models can be explored. From the hardware point of view, the advances in the new technology CUDA™ with the new boards Tesla and Fermi open new possibilities for going on with the research.

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. HC acknowledges the support of the project MTM2009-12716 of the Ministerio de Educación y Ciencia of Spain and the project PO6-TIC-02268 of Excellence of Junta de Andalucía, and the "CATAM" PAICYT research group FQM-296.

References

1. 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., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5957, pp. 227–241. Springer (2009)
2. 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)
3. 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)
4. Ceterchi, R., Gramatovici, R., Jonoska, N., Subramanian, K.G.: Tissue-like P systems with active membranes for picture generation. Fundamenta Informaticae 56(4), 311–328 (2003)
5. Ceterchi, R., Mutyam, M., Păun, G., Subramanian, K.G.: Array-rewriting P systems. Natural Computing 2(3), 229–249 (2003)
6. Chao, J., Nakayama, J.: Cubical singular simplex model for 3D objects and fast computation of homology groups. In: 13th International Conference on Pattern Recognition (ICPR'96). vol. IV, pp. 190–194. IEEE Computer Society, IEEE Computer Society, Los Alamitos, CA, USA (1996)
7. 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 (ROMJIST) 13(2), 131–140 (2010)

8. 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.) CIARP. Lecture Notes in Computer Science, vol. 5856, pp. 169–176. Springer (2009)
9. Christinal, H.A., Díaz-Pernil, D., Real, P.: Using membrane computing for obtaining homology groups of binary 2D digital images. In: Wiederhold, P., Barneva, R.P. (eds.) IWCIA. Lecture Notes in Computer Science, vol. 5852, pp. 383–396. Springer (2009)
10. Christinal, H.A., Díaz-Pernil, D., Real, P.: P systems and computational algebraic topology. Journal of Mathematical and Computer Modelling 52(11-12), 1982 – 1996 (December 2010), the BIC-TA 2009 Special Issue, International Conference on Bio-Inspired Computing: Theory and Applications
11. 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 (2010)
12. 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. Theoretical Computer Science 404(1-2), 76–87 (2008)
13. Díaz-Pernil, D., Pérez-Jiménez, M.J., Romero, A.: Efficient simulation of tissue-like P systems by transition cell-like P systems. Natural Computing 8, 797–806 (2009)
14. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON. Lecture Notes in Computer Science, vol. 2387, pp. 290–299. Springer (2002)
15. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
16. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with cuda. Queue 6, 40–53 (March 2008)
17. 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 (May 2008)
18. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1), 80–113 (2007)
19. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin, Germany (2002)
20. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
21. Ritter, G.X., Wilson, J.N., Davidson, J.L.: Image algebra: An overview. Computer Vision, Graphics, and Image Processing 49(3), 297–331 (1990)
22. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
23. NVIDIA Corporation. NVIDIA CUDA^{tm} Programming Guide. `http://www.nvidia.com/object/cuda_home_new.html`
24. P system web page. http://ppage.psystems.eu

Elementary Active Membranes Have the Power of Counting

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron

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

Summary. We prove that uniform families of P systems with active membranes operating in polynomial time can solve the whole class of **PP** decision problems, without using nonelementary membrane division or dissolution rules. This result also holds for families having a stricter uniformity condition than the usual one.

1 Introduction

P systems with active membranes [9] are known to solve computationally hard problems in polynomial time by trading space for time: an exponential number of membranes is created in polynomial time by using division rules, and then massive parallelism is exploited, e.g., to explore the whole solution space of an **NP**-complete problem in parallel.

When we allow nonelementary division rules, i.e., rules that can be applied to membranes containing further membranes, even **PSPACE**-complete problems become solvable in polynomial time [10, 2]. The general idea is that nonelementary division allows us to construct a binary tree-shaped membrane structure, isomorphic to the parse tree of the formula resulting from the expansion of universal and existential quantifiers into conjunctions and disjunctions, according to the equivalences

$$\forall x\, \varphi(x) \Leftrightarrow \varphi(0) \wedge \varphi(1) \qquad\qquad \exists x\, \varphi(x) \Leftrightarrow \varphi(0) \vee \varphi(1).$$

We also know that no problem outside **PSPACE** can be solved in polynomial time, as this is also an upper bound [11]: in symbols, we have $\mathbf{PMC}_{\mathcal{AM}} = \mathbf{PSPACE}$.

On the other hand, when no division at all is allowed the resulting P systems can be shown to be no more powerful than polynomial-time Turing machines ("Milano Theorem" [12]).

The "intermediate" case, when the only membranes that can divide are elementary (i.e., leaves of the tree corresponding to the membrane structure), is possibly

the most interesting one. The exponential number of membranes that may be created cannot be structured into a binary tree: hence, the algorithm above can only be applied to formulae having just one kind of quantifier. This is enough to solve the SAT problem [12] and its complement (which are respectively **NP**- and **coNP**-complete), where only existentially (resp., universally) quantified variables are allowed.[1] However, the corresponding complexity class $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ still lacks a characterisation in terms of Turing machines. Alhazov et al. [1] have shown how **PP**- and #**P**-complete problems can be solved without nonelementary division, but their result is not directly related to the class $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$, as it requires some form of post-processing or the use of non-standard rules. In this paper, we improve the previous $\mathbf{NP} \cup \mathbf{coNP}$ lower bound to **PP** within the standard framework of active membranes. This is an improved version of the paper "P systems with active membranes: Beyond NP and coNP" presented by the authors and the Eleventh International Conference on Membrane Computing [7].

2 Preliminaries

We use P systems with restricted elementary active membranes, which are defined as follows.

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

- Γ *is a finite alphabet of symbols (the objects);*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree) consisting of* d *membranes enumerated by* $1, \ldots, d$*; furthermore, each membrane is labeled by an element of* Λ*, not necessarily in a one-to-one way;*
- $w_1, \ldots, w_d$ *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).

[1] Some further partial results relating quantifier alternations and nonelementary division depth, albeit in the slightly different framework of P systems with active membranes without charges, have been obtained [8].

- *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.
- *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.

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 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 nonelementary 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 we proceed 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.

A *halting computation* of Π 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 *recognisers* 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.

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

Definition 2. *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 with a distinguished input membrane (the P systems structure Π_n is common for all inputs of length n).*
- *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.*

Notice that this definition of uniformity is possibly weaker than the other one commonly used in membrane computing [6], 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 $|x|$). In particular, complexity classes defined using this restricted uniformity condition are not always formally known to be closed under polynomial-time reductions[2]. See [4] for further details on uniformity conditions, including constructions using weaker devices than polynomial-time Turing machines.

Any explicit encoding of Π_x is allowed as output, 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

[2] This might complicate proofs of inclusions among complexity classes, although one can usually find a proof not relying on closure under polynomial-time reductions, as in the present paper.

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 in proportion to their number. For instance, the membrane structure can be represented by brackets, and the multisets as strings (i.e., in unary notation); this is a *permissible encoding* in the sense of [4].

Finally, we describe how time complexity for families of recogniser P systems is measured.

Definition 3. *A uniform family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to decide the language* $L \subseteq \Sigma^\star$ *(in symbols* $L(\boldsymbol{\Pi}) = L$*) in time* $f\colon \mathbb{N} \to \mathbb{N}$ *iff, for each* $x \in \Sigma^\star$*,*

- *the system* Π_x *accepts if* $x \in L$*, and rejects if* $x \notin L$*;*
- *each computation of* Π_x *halts within* $f(|x|)$ *computation steps.*

In this paper we use uniform families of P systems to solve a variant of the SAT problem. Hence, we set the relevant notation and describe how Boolean formulae can be encoded in order to simplify a uniform solution.

Given a set of $m \geq 3$ variables $X_m = \{x_1, \ldots, x_m\}$, the number of clauses of 3 variables (without repeated variables, and ignoring permutations of literals) is given by $8\binom{m}{3}$, the number of 3-element subsets times the 2^3 ways to negate them. Hence, a 3CNF formula φ can be encoded as an $8\binom{m}{3}$-bit string, where the i-th bit is 1 iff the i-th clause (under some fixed ordering) appears in φ. Notice that $8\binom{m}{3} = \frac{4}{3}m^3 - 4m^2 + \frac{8}{3}m$ is a polynomial.

In this paper, we will order the clauses according to the enumeration printed by the recursive algorithm of Figure 1.

Example 1 (Encoding). The clauses over 4 variables $X_4 = \{x_1, \ldots, x_4\}$, in the order given by the algorithm above, are the following ones:

$x_1 \vee x_2 \vee x_3$	$x_1 \vee x_2 \vee \overline{x}_3$	$x_1 \vee \overline{x}_2 \vee x_3$	$x_1 \vee \overline{x}_2 \vee \overline{x}_3$
$\overline{x}_1 \vee x_2 \vee x_3$	$\overline{x}_1 \vee x_2 \vee \overline{x}_3$	$\overline{x}_1 \vee \overline{x}_2 \vee x_3$	$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$
$x_1 \vee x_2 \vee x_4$	$x_1 \vee x_2 \vee \overline{x}_4$	$x_1 \vee \overline{x}_2 \vee x_4$	$x_1 \vee \overline{x}_2 \vee \overline{x}_4$
$\overline{x}_1 \vee x_2 \vee x_4$	$\overline{x}_1 \vee x_2 \vee \overline{x}_4$	$\overline{x}_1 \vee \overline{x}_2 \vee x_4$	$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4$
$x_1 \vee x_3 \vee x_4$	$x_1 \vee x_3 \vee \overline{x}_4$	$x_1 \vee \overline{x}_3 \vee x_4$	$x_1 \vee \overline{x}_3 \vee \overline{x}_4$
$\overline{x}_1 \vee x_3 \vee x_4$	$\overline{x}_1 \vee x_3 \vee \overline{x}_4$	$\overline{x}_1 \vee \overline{x}_3 \vee x_4$	$\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4$
$x_2 \vee x_3 \vee x_4$	$x_2 \vee x_3 \vee \overline{x}_4$	$x_2 \vee \overline{x}_3 \vee x_4$	$x_2 \vee \overline{x}_3 \vee \overline{x}_4$
$\overline{x}_2 \vee x_3 \vee x_4$	$\overline{x}_2 \vee x_3 \vee \overline{x}_4$	$\overline{x}_2 \vee \overline{x}_3 \vee x_4$	$\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4$

Then, the formula

$$\varphi = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4)$$

is encoded as the following sequence of $8\binom{4}{3} = 32$ bits

```
PRINT-CLAUSES(m)
    IF m > 3 THEN
        PRINT-CLAUSES(m − 1)
    END
    FOR i ← 1 TO m − 2 DO
        FOR j ← i + 1 TO m − 1 DO
            PRINT "x_i ∨ x_j ∨ x_m"
            PRINT "x_i ∨ x_j ∨ x̄_m"
            PRINT "x_i ∨ x̄_j ∨ x_m"
            PRINT "x_i ∨ x̄_j ∨ x̄_m"
            PRINT "x̄_i ∨ x_j ∨ x_m"
            PRINT "x̄_i ∨ x_j ∨ x̄_m"
            PRINT "x̄_i ∨ x̄_j ∨ x_m"
            PRINT "x̄_i ∨ x̄_j ∨ x̄_m"
        END
    END
END
```

Fig. 1. A recursive, polynomial-time algorithm that enumerates all clauses of 3 out of m variables.

$$\varphi = 0100\ 0001\ 0000\ 0000\ 0001\ 0000\ 0000\ 0010$$

because the clauses actually appearing are the 2nd, 8th, 20th, and 31st ones.

Besides being computable in polynomial time with respect to m, this ordering has the following important property: the sequence of clauses over m variables is a prefix of the sequence of clauses over m' variables whenever $m' \geq m$. As a consequence, each formula over m variables can also be considered as a formula over m' variables by padding its encoding to the correct length. For instance, the formula φ of Example 1 can be interpreted as a formula over *five* variables $x_1, \ldots, x_5$ if its encoding is padded to length $8\binom{5}{3} = 80$ by a string of zeroes, i.e., as $\varphi \cdot 0^{48}$.

We now consider the following decision problem.

Problem 1 (Threshold-3SAT). Given a Boolean formula φ over m variables and a non-negative integer $k < 2^m$, do more than k assignments (out of 2^m) satisfy it?

Notice that we can force all valid instances (φ, k) of Problem 1 to have a description of length exactly $8\binom{m}{3} + m$ for some m, as every number in the range $[0, 2^m)$ can be represented using m bits. This will be useful in the next section.

Proposition 1. THRESHOLD-3SAT *is* **PP**-*hard.*

Proof. We reduce the following standard **PP**-complete problem [5, p. 256] to THRESHOLD-3SAT.

Problem 2 (Majority-SAT). Given a Boolean formula φ in CNF, having c clauses over m variables and such that each variable occurs at most once per clause, do more than half the assignments (i.e., more than 2^{m-1} assignments) satisfy it?

The reduction is similar to that from SAT to 3SAT described in [3, p. 48]. We first transform φ into a formula having *at most* three literals per clause. Observe that φ is satisfied iff the formula obtained by replacing a clause of $p > 3$ literals $\bigvee_{i=1}^{p} \ell_i$ with

$$(y \Leftrightarrow \ell_1 \vee \ell_2) \wedge \left(y \vee \bigvee_{i=3}^{p} \ell_i \right)$$

is also satisfied, assuming y is a new variable. In CNF, that is equivalent to

$$(\bar{\ell}_1 \vee y) \wedge (\bar{\ell}_2 \vee y) \wedge (\ell_1 \vee \ell_2 \vee \overline{y}) \wedge \left(y \vee \bigvee_{i=3}^{p} \ell_i \right).$$

This substitution doubles the number of total assignments of the formula, due to the addition of a new variable, but the number of *satisfying* ones is left unchanged, as the value of y is forced to be equal to $\ell_1 \vee \ell_2$. The substitution decreases by one the number of literals of the initial clause; by repeating the process $p - 3$ times, and then again to any other clause having more than three literals, we obtain a formula φ' having at most three literals per clause, and the same number of satisfying assignments as φ. The number of variables of φ' is bounded by $m + cm$.

Next, we transform every clause of one or two literals into a clause of exactly three. A clause of a single literal ℓ is replaced by

$$(\ell \vee z_1 \vee z_2) \wedge (\ell \vee \overline{z}_1 \vee z_2) \wedge (\ell \vee z_1 \vee \overline{z}_2) \wedge (\ell \vee \overline{z}_1 \vee \overline{z}_2),$$

where z_1 and z_2 are new variables, which is clearly satisfied iff ℓ is. Each replacement like this one multiplies by $2^2 = 4$ the number of satisfying assignments of the whole formula, as the values of z_1 and z_2 are actually irrelevant.

A clause of two literals $\ell_1 \vee \ell_2$ is replaced by

$$(\ell_1 \vee \ell_2 \vee z) \wedge (\ell_1 \vee \ell_2 \vee \overline{z}),$$

where z is a new variable, which is also equivalent to the original clause but doubles the number of satisfying assignments of the formula.

Call φ'' the formula obtained from φ' by replacing single and 2-literal clauses by conjunctions of 3-literal clauses as described above, and let q be the number of variables added in the process (notice that q is $O(cm)$). Then it should be clear that φ has more than 2^{m-1} satisfying assignments iff φ' does, and the latter is equivalent to φ'' having more than 2^{m+q-1} satisfying assignment.

Since the mapping $R(\varphi) = (\varphi'', 2^{m+q-1})$ is computable in polynomial time with respect to c and m, it is a reduction from Majority-SAT to Threshold-3SAT. □

3 Solving Threshold-3SAT

In order to solve Threshold-3SAT we design a polynomial time, deterministic Turing machine F (for "family") such that, for each n of the form $8\binom{m}{3} + m$, the output of $F(1^n)$ is a P system Π_n that solves the problem for all inputs of length n.

The input provided to Π_n is computed by another polynomial time Turing machine E (for "encoding") that, given an m-variable 3CNF formula as described in the previous section and an integer k, outputs the following set of objects:

$$\begin{aligned} E(\varphi, k) = \{&\mathrm{C}_i : \text{the } i\text{-th clause does } \textit{not} \text{ appear in } \varphi, \text{ for } 1 \leq i \leq 8\tbinom{m}{3}\} \cup \\ \{&\mathrm{K}_i : \text{the } i\text{-th bit of } k \text{ (counting from 0) is 1, for } 1 \leq i \leq m-1\} \end{aligned}$$

Example 2. The formula φ of Example 1, together with the integer $k = 12$, are encoded as $E(\varphi, k) = \{\mathrm{C}_i : 1 \leq i \leq 32 \text{ and } i \notin \{2, 8, 20, 31\}\} \cup \{\mathrm{K}_2, \mathrm{K}_3\}$.

The initial configuration of Π_n, input multiset excluded, is the following one:

$$\mathcal{C}_0 = \left[\left[\mathrm{I}_{n-m}\right]^0_{\mathrm{E}} \left[\ \right]^0_{\mathrm{K}0} \cdots \left[\ \right]^0_{\mathrm{K}m-1} \mathrm{O}_{t+1}\ \mathrm{NO}_{t+3}\right]^0_{\mathrm{IN}}$$

where $t = 4n - 3m + 4$. The multiset encoding $E(\varphi, k)$ is placed inside the input membrane in, and then the computation proceeds according to the following five phases:

1. Initialise the contents of the membranes.
2. Generate all possible assignments for φ.
3. Check if each assignment satisfies the input formula φ.
4. Count the number of assignments, testing whether it is larger than k.
5. Output the correct answer.

The fourth phase, first suggested by Alhazov et al. [1], differentiates our solution from the standard algorithm schema, common in membrane computing, for solving **NP**-complete problems.[3]

Phase 1 (Initialise). In the first computation steps, the objects C_i, corresponding to the clauses that do not appear in the input formula φ, are moved to membrane E using the communication rules

$$\mathrm{C}_i\ [\ ]^0_{\mathrm{E}} \to [\mathrm{C}_i]^0_{\mathrm{E}} \qquad \text{for } 1 \leq i \leq n - m. \tag{R_1}$$

This takes a number of steps at most equal to $n-m$ (i.e., to the maximum number of clauses in φ). In the mean time, the object I_{n-m} has its subscript decreased by one for $n-m-1$ computation steps, and is finally replaced during the $(n-m)$-th step, using the rules

[3] Indeed, by eliminating the fourth phase (or, equivalently, by choosing $k = 0$) we obtain essentially a uniform version of the original solution to SAT described by Zandron et al. [12].

$$[\mathrm{I}_i \to \mathrm{I}_{i-1}]^0_{\mathrm{E}} \qquad \text{for } 1 \le i \le n-m. \tag{R_2}$$

$$[\mathrm{I}_0 \to \mathrm{X}_1 \cdots \mathrm{X}_m \mathrm{W}_m]^0_{\mathrm{E}} \tag{R_3}$$

Hence, after $n-m$ computation steps, membrane E contains C_i for each missing clause, and the variable-objects $\mathrm{X}_1, \ldots, \mathrm{X}_m$.

At the same time, the objects K_i are first moved to their respective membranes in the first time step, making them positively charged

$$\mathrm{K}_i\, [\,]^0_{\mathrm{K}_i} \to [\mathrm{K}_i]^+_{\mathrm{K}_i} \qquad \text{for } 0 \le i \le m-1 \tag{R_4}$$

then each K_i divides its membrane i times:

$$[\mathrm{K}_i]^+_{\mathrm{K}_j} \to [\mathrm{K}_{i-1}]^+_{\mathrm{K}_j}\, [\mathrm{K}_{i-1}]^+_{\mathrm{K}_j} \qquad \text{for } 0 \le j \le m-1 \text{ and } 1 \le i \le j. \tag{R_5}$$

After at most m steps (the largest possible subscript is $m-1$), there are exactly k positively charged membranes among those having label $\mathrm{K}_0, \ldots, \mathrm{K}_{m-1}$.

The total duration of Phase 1 is $n-m$ steps.

Phase 2 (Generate). The variable-objects $\mathrm{X}_1, \ldots, \mathrm{X}_m$ are used to generate all the truth assignment inside multiple copies of membrane E. This is accomplished by using the division rules

$$[\mathrm{X}_i]^0_{\mathrm{E}} \to [\mathrm{T}_i]^0_{\mathrm{E}}\, [\mathrm{F}_i]^0_{\mathrm{E}} \qquad \text{for } 1 \le i \le m. \tag{R_6}$$

After m steps, we have 2^m copies of membrane E, each one containing a different truth assignment to the variables $x_1, \ldots, x_m$ of φ: the occurrence of T_i (resp., F_i) indicates that x_i is set to true (resp., false) in that particular assignment.

Simultaneously, the subscript of object W_m (standing for "wait m steps") is decreased by one each step:

$$[\mathrm{W}_i \to \mathrm{W}_{i-1}]^0_{\mathrm{E}} \qquad \text{for } 1 \le i \le m. \tag{R_7}$$

When the counter reaches 0, the objects W_0 are sent out from each copy of membrane E while changing its charge according to the following rule:

$$[\mathrm{W}_0]^0_{\mathrm{E}} \to [\,]^+_{\mathrm{E}}\, \mathrm{W}_0. \tag{R_8}$$

When membrane E is positively charged, the objects T_i and F_i are replaced by the set of clause-objects corresponding to all the clauses satisfied by that particular value of variable x_i (whether they are actually part of formula φ or not):

$$[\mathrm{T}_i \to \mathrm{C}_{i_1} \cdots \mathrm{C}_{i_\ell}]^+_{\mathrm{E}} \quad \text{for } 1 \le i \le m, \text{ where clause } i_j \text{ contains literal } x_i \tag{R_9}$$

$$[\mathrm{F}_i \to \mathrm{C}_{i_1} \cdots \mathrm{C}_{i_\ell}]^+_{\mathrm{E}} \quad \text{for } 1 \le i \le m, \text{ where clause } i_j \text{ contains literal } \bar{x}_i. \tag{R_{10}}$$

Notice that $\ell = 4\binom{m-1}{2} = 2m^2 - 6m + 4$, as this is the number of clauses over m variables where a particular literal occurs.

At the same time, each copy of W_0 is brought back as S_0 to a copy of membrane E by using the following rule in a maximally parallel way:

$$\mathrm{W}_0\, [\,]^+_{\mathrm{E}} \to [\mathrm{S}_0]^+_{\mathrm{E}}. \tag{R_{11}}$$

The total duration of Phase 2 is $m+2$ steps.

Phase 3 (Check). The occurrence of S_i inside a copy of membrane E denotes the fact that the first i clauses (according to the enumeration described above) have been found to be satisfied by the assignment corresponding to that membrane. We assume that the clauses which do *not* appear in φ are satisfied by default; indeed, this is precisely the reason why the corresponding C_i objects were placed inside membrane E in Phase 1.

When membrane E is positively charged, the object C_1 is sent out from E (as the "junk" object #), changing the charge to negative:

$$[\mathrm{C}_1]^+_{\mathrm{E}} \to [\,]^-_{\mathrm{E}}\ \#. \tag{R_{12}}$$

When E is negative, object S_i is sent out; at the same time, the objects C_i, for $i \geq 2$, are temporarily "primed", and all remaining copies of C_1 are discarded:

$$[\mathrm{S}_i]^-_{\mathrm{E}} \to [\,]^-_{\mathrm{E}}\ \mathrm{S}_i \qquad \text{for } 0 \leq i \leq n-m-1 \tag{R_{13}}$$

$$[\mathrm{C}_i \to \mathrm{C}'_i]^-_{\mathrm{E}} \qquad \text{for } 2 \leq i \leq n-m \tag{R_{14}}$$

$$[\mathrm{C}_1 \to \#]^-_{\mathrm{E}}. \tag{R_{15}}$$

In the next step, the objects C'_i become C_{i-1}; this way, during this phase we only need to check for the presence of object C_1 for $n-m$ times.

$$[\mathrm{C}'_i \to \mathrm{C}_{i-1}]^-_{\mathrm{E}} \qquad \text{for } 2 \leq i \leq n-m. \tag{R_{16}}$$

At the same time, the object S_i is brought back in (if $i < n-m$), its subscript incremented by one, while changing the charge of E to positive in order to resume the checking of clauses:

$$\mathrm{S}_i\ [\,]^-_{\mathrm{E}} \to [\mathrm{S}_{i+1}]^+_{\mathrm{E}} \qquad \text{for } 0 \leq i \leq n-m-1. \tag{R_{17}}$$

If S_{n-m} is finally found inside E, it is sent out to signal that the formula is fully satisfied under that particular assignment:

$$[\mathrm{S}_{n-m}]^+_{\mathrm{E}} \to [\,]^0_{\mathrm{E}}\ \mathrm{S}_{n-m}. \tag{R_{18}}$$

Hence, after the $3n-3m+1$ steps of Phase 3, the outermost membrane IN contains a copy of S_{n-m} for each assignment that satisfies φ.

Phase 4 (Count). In the next step, k copies of S_{n-m} (or all of them, if less than k exist) are "deleted" from membrane IN by sending them into any of the membranes having label $\mathrm{K}_0, \ldots, \mathrm{K}_{m-1}$; these membranes are set to negative in the process, to avoid absorbing multiple objects:

$$\mathrm{S}_{n-m}\ [\,]^+_{\mathrm{K}_i} \to [\#]^-_{\mathrm{K}_i} \qquad \text{for } 0 \leq i \leq m-1. \tag{R_{19}}$$

Recall that the number of positively charged membrane K_i is exactly k. Hence, after this single step there are one or more copies of S_{n-m} left inside membrane IN if and only if the number of satisfying assignments of φ was greater than k.

Phase 5 (Output). The objects o_t and NO_{t+2}, initially located inside membrane IN, work as counters during Phases 1–4 (whose total duration is precisely $t = 4n - 3m + 4$ steps) according to the following rules:

$$[\mathrm{o}_i \to \mathrm{o}_{i-1}]^0_{\mathrm{IN}} \qquad \text{for } 1 \le i \le t+1 \qquad (R_{20})$$

$$[\mathrm{NO}_i \to \mathrm{NO}_{i-1}]^0_{\mathrm{IN}} \qquad \text{for } 2 \le i \le t+3. \qquad (R_{21})$$

When the subscript of o_i reaches 0, Phase 5 begins and o_0 is sent out, thus "opening" membrane IN for output by setting its charge to positive:

$$[\mathrm{o}_0]^0_{\mathrm{IN}} \to [\,]^+_{\mathrm{IN}} \#. \qquad (R_{22})$$

If any object s_{n-m} is found inside IN, it is sent out as YES in the next step, changing the charge to negative:

$$[\mathrm{s}_{n-m}]^+_{\mathrm{IN}} \to [\,]^-_{\mathrm{IN}}\ \mathrm{YES}. \qquad (R_{23})$$

Otherwise, membrane IN remains positive, and the object NO_0, produced by the rule

$$[\mathrm{NO}_1 \to \mathrm{NO}_0]^+_{\mathrm{IN}} \qquad (R_{24})$$

is sent out as NO in the following step:

$$[\mathrm{NO}_0]^+_{\mathrm{IN}} \to [\,]^-_{\mathrm{IN}}\ \mathrm{NO}. \qquad (R_{25})$$

The duration of Phase 5 is either 2 or 3 steps, depending on whether the number of assignments satisfying φ is greater than k or not.

This algorithm allows us to solve the Threshold-3SAT problem in linear time $O(n+m) = O(n)$.

Theorem 1. Threshold-3SAT $\in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$.

Proof. Let $\boldsymbol{\Pi}$ be the family of P systems described above. We show that $\boldsymbol{\Pi}$ can be constructed in polynomial time by two Turing machines F and E.

The machine F, on input 1^n (where $n = |(\varphi, k)|$) first computes the unique positive root of the polynomial

$$p(m) = 8\tbinom{m}{3} + m - n$$

thus establishing the number of variables. This can be done in polynomial time with respect to n simply by trying all integers up to n.

Then F outputs the initial configuration $\mathcal{C}_0$ of Π_n, which can be easily computed in polynomial time from n and m. Finally, the set of rules $R_1 \cup \cdots \cup R_{25}$ is output. Each of the sets R_i can be computed in polynomial time; the most complicated ones are R_9 and R_{10}, which require enumerating the clauses using the algorithm of Figure 1.

The P system Π_n itself, on input $E(\varphi, k)$, only requires $O(n)$ time (and $O(n2^m)$ space) to output YES or NO, without using any nonelementary division or dissolution rules; this establishes that the problem is in $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$.

If the machines F and E receive a malformed input, i.e., any input having length $n \neq 8\binom{m}{3} + m$ for all $m \leq n$, then F produces a fixed P system that sends out the NO object immediately (while E produces an empty multiset). □

4 Solving the other PP problems

Being able to solve one **PP**-complete problem implies $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ if the uniformity condition is defined as in [6], as closure under polynomial-time reductions is immediate. However, our uniformity condition is possibly weaker, as the P system associated with each input only depends on its size and not on the specific input itself, and the class $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ defined this way is currently not known to be closed under polynomial-time reductions. Hence, to prove the **PP** inclusion we operate as follows.

Theorem 2. $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$.

Proof. Let $L \in \mathbf{PP}$, and let R be a Turing machine reducing L to the problem THRESHOLD-3SAT in polynomial time $p(n)$, where n is the length of the instance of L. We describe two polynomial-time Turing machines F' and E' constructing a family of P systems $\boldsymbol{\Pi}'$, also running in polynomial time, such that $L(\boldsymbol{\Pi}') = L$.

The machine F', on input 1^n (where $n = |x|$), constructs a P system able to solve the largest THRESHOLD-3SAT formula that might be produced as the output of R; if the actual output of R is smaller than that, we can pad it to the correct length by adding enough zeroes. Let f be defined as follows:

$$f(n) = \min\left\{n' : n' \geq n \text{ and } n' = 8\tbinom{m'}{3} + m' \text{ for some } m'\right\}$$

that is, $f(n)$ is the smallest integer of the form $8\binom{m'}{3} + m'$ greater than or equal to n. Then, F' behaves as follows:

$$F'(1^n) = F\left(1^{f(p(n))}\right) = \Pi_{f(p(n))}.$$

Since R runs in time $p(n)$, the P system $\Pi_{f(p(n))}$ is large enough to receive as input any formula φ obtained via the reduction R, as $|R(x)| = |(\varphi, k)| \leq p(|x|)$, as long as it is padded to length $f(p(n))$ as described above.

Notice that the value $f(n)$ can be obtained in polynomial time with respect to n by simply computing $8\binom{m'}{3} + m'$ for all integers m' until n is reached or exceeded; furthermore, $f(n)$ itself is at most polynomial in n (e.g., a trivial upper bound is $8\binom{n}{3} + n$).

The encoding machine E', on input x, produces an output formula encoding φ', obtained from $(\varphi, k) = R(x)$ as follows:

$$\varphi' = \varphi \cdot 0^{\ell} \qquad \text{where } \ell = f(p(n)) - |\varphi|.$$

Recall from Section 2 that $\varphi \cdot 0^{\ell}$ is indeed a valid encoding of a formula, having exactly the same clauses of φ *but over* m' *variables instead of the original* m (where m' satisfies $f(p(n)) = 8\binom{m'}{3} + m'$). The number of required assignments k has to be adjusted accordingly: every assignment of the original formula φ corresponds to $2^{m'-m}$ assignments of φ' (obtained by extending it with arbitrary values to the new variables) that satisfy it iff the original assignment satisfies φ, since the new $m' - m$ variables do not actually appear in φ'. Hence, we define $k' = 2^{m'-m} \cdot k$.

Summarising, the machine E' behaves as follows:

$$E'(x) = E\big(\varphi \cdot 0^{\ell}, 2^{m'-m}k\big) \qquad \text{where } (\varphi, k) = R(x).$$

Since $(\varphi', k') \in$ Threshold-3SAT iff $(\varphi, k) \in$ Threshold-3SAT by construction, and the latter is equivalent to $x \in L$ by reduction, we obtain $L \in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$. But L was an arbitrary **PP** language: hence the inclusion $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ holds as required. □

5 Conclusions

Uniform families of P systems with active membranes without nonelementary division and dissolution rules have been proved to be able to solve all **PP** problems in polynomial time; this property holds even when the uniformity condition is the same as that used for traditional families of circuits. The current bounds on the computing power of these P systems, in terms of complexity classes for Turing machines, are thus

$$\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})} \subseteq \mathbf{PSPACE} = \mathbf{PMC}_{\mathcal{AM}},$$

where neither inclusion is known to be proper. Further improvements on the **PP** lower bound are expected, as it is plausible that P systems like those of the family $\boldsymbol{\Pi}$ solving Threshold-3SAT described in this paper can be used as "modules" in larger P systems, thus providing a way to simulate an oracle for a **PP**-complete problem. Furthermore, it is still possible that $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ actually coincides with **PSPACE**, thus showing that nonelementary membrane division (and possibly dissolution) do not increase the efficiency of P systems with active membranes.

References

1. Alhazov, A., Burtseva, L., Cojocaru, S., Rogozhin, Y.: Solving PP-complete and #P-complete problems by P systems with active membranes. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing 9th International Workshop, WMC 2008, Lecture Notes in Computer Science, vol. 5931, pp. 108–117. Springer (2009)

2. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundamenta Informaticae 58(2), 67–77 (2003)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1979)
4. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
5. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
6. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
7. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with elementary active membranes: Beyond NP and coNP. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 11th International Conference, CMC 2010, Lecture Notes in Computer Science, vol. 6501, pp. 338–347. Springer (2011)
8. Porreca, A.E., Murphy, N.: First steps towards linking membrane depth and the polynomial hierarchy. In: Martínez-del-Amor, M.A., Păun, G., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Eight Brainstorming Week on Membrane Computing, RGNC Reports, vol. 1/2010, pp. 255–266. Fénix Editora (2010)
9. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
10. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
11. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
12. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)

Integer Linear Programming for Tissue-like P Systems

Raúl Reina-Molina[1], Daniel Díaz-Pernil[1], Miguel A. Gutiérrez-Naranjo[2]

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

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

Summary. In this paper we report a work-in-progress whose final target is the implementation of tissue-like P system in a cluster of computers which solves some instances of the segmentation problem in 2D Digital Imagery. We focus on the theoretical aspects and the problem of choosing a maximal number of application of rules by using Integer Linear Programming techniques. This study is on the basis of a future distribution of the parallel work among the processors.

1 Introduction

Membrane systems[1] are distributed and parallel computing devices processing multisets of objects in compartments delimited by membranes. Computation is carried out by applying given rules to every membrane content, usually in a maximal non-deterministic way, although other semantics are being explored.

In spite of some recent efforts (see [6]), there are neither *in vivo*, *in vitro* nor *in silico* implementations of such devices and the unique way to get a mechanical application of the rules is by the development of software tools capable of performing simulations of such devices on current computers [4, 7].

In this paper we report a work-in-progress about the implementation of tissue-like P system in a cluster of computers. This is not the first attempt. In 2003, Ciobanu and Wenyuan presented in [3], a parallel implementation of transition P systems. The program was designed for a cluster of 64 dual processor nodes and

[1] We refer to [13] for basic information in this area, to [14] for a comprehensive presentation and the web site [15] for the up-to-date information.

it was implemented and tested on a Linux cluster at the National University of Singapore.

We will focus on the problem of finding a maximal amount of applications of rules from a given configuration in the framework of tissue-like P systems. The contribution of this paper is the use of a matrix representation for configuratios and rules and the use of Integer Linear Programming.

In this paper we will consider a matrix representation of tissue-like P systems. This new representation will be useful for considering Integer Linear Programming for automatically searching a maximal set of rules.

We also start using Operation Research techniques for calculating the rules to be applied in each computing step.

A linear program, LP for short, is an Operation Research problem consisting in optimizing a linear function subject to linear restrictions. Without lost of generality we may assume that a LP is a problem like the following:

$$(LP)\begin{cases} \text{maximize: } \displaystyle\sum_{k=1}^{n} c_k x_k \\ \text{subject to:} \\ \qquad \displaystyle\sum_{k=1}^{n} a_{1k} x_k \leq b_1 \\ \qquad \cdots \\ \qquad \displaystyle\sum_{k=1}^{n} a_{mk} x_k \leq b_m \end{cases}$$

When the additional constraint of integrality of x_k, the LP is called Integer Linear Program, ILP for short.

The paper is organized as follows: First we briefly recall some basic definitions related to multisets and tissue-like P systems. Next, we show a theoretical study on how the tissue-like P systems can adopt a matrix representation. We show that this representation can be useful for using Integer Linear Programming for finding a maximal set of applications which will be used for a future distribution of the work among different processors. We illustrates this definition with an explicative example. Finally, some clues for the future work are presented.

2 Preliminaries

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

A *multiset over a set* A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$ and its *size* is defined as $\sum_{x \in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over A, then it will be denoted as $m = a_1^{f(a_1)} a_2^{f(a_2)} \cdots a_k^{f(a_k)}$ or $\{\{a_j^{f(a_j)} ; 1 \leq j \leq k\}\}$ where $supp(m) = \{a_1, \ldots, a_k\}$, and for each element a_i, $f(a_i)$ is called the multiplicity of a_i. furthermore the multiplicity of an element $a_i \in m$ is denoted as $\text{mult}(a_i, m)$. In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For details, see [14].

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

In *tissue P systems*, the tree-like membrane structure is replaced by a general graph. This model has two biological inspirations (see [9, 10]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules. In symport rules, objects cooperate to traverse a membrane together in the same direction, whereas in the case of antiport rules, objects residing at both sides of the membrane cross it simultaneously but in opposite directions.

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

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

where

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

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

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

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

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

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

3 Encoding Tissue-like P Systems by Using Matrices

In this section we define the formal framework for a new way of calculating maximal set of rules to be applied in tissue-like P systems. First of all let us suppose that we have the alphabet indexed, so $\Gamma = \{\gamma_j : 1 \leq j \leq |\Gamma|\}$. In the same sense let $\mathcal{R} = \{r_k : 1 \leq k \leq |\mathcal{R}|\}$ be the set of communication rules. By using the order in Γ settled by the indexation, we can consider the *vector representation* [1], $\underline{u} \in \mathbb{N}^{|\Gamma|}$ of the multiset u as the $|\Gamma|$-dimensional vector $\underline{u}$ with $\underline{u}_j = \text{mult}(\gamma_j, u)$ for each $j = 1, 2, \ldots, |\Gamma|$. Moreover, for technical reasons, we will extend this vectorial representation to the environment by including the symbol ∞ for the objects with an arbitrary amount of copies. In this way, we will consider a vector $\underline{u}$ with coordinates in $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ with

$$\underline{u}_j = \begin{cases} \text{mult}(\gamma_j, u) \text{ if } \gamma_j \in \mathcal{E} \\ \infty \text{ if } \gamma_j \in \mathcal{E} \end{cases}$$

We can extend elementary operations in $\mathbb{N}$ to $\mathbb{N}_\infty$ with

$$\infty \pm n = \infty, \forall n \in \mathbb{N}$$
$$\infty \cdot n = \infty, \forall n \in \mathbb{N}, n \neq 0$$

By using this extension to $\mathbb{N}_\infty$, we can use a vector representation for the multisets inside the cell as as well as the multiset in the environment in each configuration.

The *configuration matrix* is the $(q+1) \times |\Gamma|$ matrix of non-negative integers whose i-th row is the vector representation of the multiset w_i. Let us recall that the rows is indexed from 0, to take in count the multiset for environment.

In the following, we do not lose generality if we consider the communication rule $(i, u/v, j)$ written with $i < j$.

Let $r = (i_u, u/v, i_v)$ be a communication rule interchanging the elements in u with the elements in v. From this characterizaction we define two matrices:

$$M_r^- = \begin{pmatrix} \vdots \\ i_u : \underline{u} \\ \vdots \\ i_v : \underline{v} \\ \vdots \end{pmatrix}, M_r^+ = \begin{pmatrix} \vdots \\ i_u : \underline{v} \\ \vdots \\ i_v : \underline{u} \\ \vdots \end{pmatrix} \tag{1}$$

for $0 \leq i \leq q, 1 \leq j \leq |\Gamma|$, where M_r^- has all the rows $\underline{0} \in \mathbb{N}^{|\Gamma|}$ except the i_u-th and i_v-th, which are, respectively, $\underline{u}$ and $\underline{v}$, and so on. Both matrices defined above makes the *matrix representation* for rule r, $M(r) = \langle M_r^-, M_r^+ \rangle$.

For example, let $\Gamma = \{a, b, c, d\}$ be an alphabet with environment $\mathcal{E} = \{c, d\}$. The multiset $u = \{\{a^2, c, d^3\}\}$ is encoded by $\underline{u} = (2, 0, 1, 3)$. The rule $r = (1, ab^2/c, 0)$ in a tissue-like P system with three cells (and the environment) is encoded by

$$M_r^- = \begin{pmatrix} 0 : (0,0,1,0) \\ 1 : (1,2,0,0) \\ 2 : (0,0,0,0) \\ 3 : (0,0,0,0) \end{pmatrix}, M_r^+ = \begin{pmatrix} 0 : (1,2,0,0) \\ 1 : (0,0,1,0) \\ 2 : (0,0,0,0) \\ 3 : (0,0,0,0) \end{pmatrix}$$

If we have a configuration matrix given by, for example,

$$M = \begin{pmatrix} 0 : (0,0,\infty,\infty) \\ 1 : (3,2,0,0) \\ 2 : (0,0,1,0) \\ 3 : (0,1,0,3) \end{pmatrix}$$

then the application of rule r gives the configuration matrix given by

$$M' = M + M_r^+ - M_r^- =$$

$$\begin{pmatrix} 0 : (0,0,\infty,\infty) \\ 1 : (3,2,0,0) \\ 2 : (0,0,1,0) \\ 3 : (0,1,0,3) \end{pmatrix} + \begin{pmatrix} 0 : (1,2,0,0) \\ 1 : (0,0,1,0) \\ 2 : (0,0,0,0) \\ 3 : (0,0,0,0) \end{pmatrix} - \begin{pmatrix} 0 : (0,0,1,0) \\ 1 : (1,2,0,0) \\ 2 : (0,0,0,0) \\ 3 : (0,0,0,0) \end{pmatrix} =$$

$$\begin{pmatrix} 0 : (1,2,\infty,\infty) \\ 1 : (2,0,1,0) \\ 2 : (0,0,1,0) \\ 3 : (0,1,0,3) \end{pmatrix}$$

If a computation step consists on applying the rule r_k for m_k times, $1 \leq k \leq |\mathcal{R}|$, M is the configuration matrix for a given configuration of the system and M_k^-, M_k^+ are the matrices defined in Equation 1 for rule r_k then

$$M + \sum_{k=1}^{|\mathcal{R}|} m_k (M_k^+ - M_k^-)$$

is the configuration matrix for the configuration after the computation step. Hence, if $\mathbb{M}$ is a multiset of rules, the result of applying all the rules in $\mathbb{M}$ to the configuration given by M, is the configuration obtained from matrix

$$M' = M + \mathbb{M} = M + \sum_{r \in \mathbb{M}} \text{mult}(r, \mathbb{M}) M_r$$

where $M_r = M_r^- + M_r^+$

A rule r can be applied if there are enough objects in each cell to be communicated. Thus, if M is a configuration matrix and $\langle M_r^-, M_r^+ \rangle$ is the matrix representation of the communication rule r, it is clear that the rule can be applied if and only if $M + M(r)^- \geq \mathbf{0}$, where $\mathbf{0}$ is the null matrix and $\geq$ is considered elementwise. Thus, if $\mathbb{M}$ is a multiset of rules, then

$$\mathbb{M} \text{ can be applied } \Leftrightarrow M + \sum_{r \in \mathbb{M}} \text{mult}(r, \mathbb{M}) M_r^- \geq \mathbf{0} \tag{2}$$

We will consider the following definition of maximality. A maximal multiset of rules is a multiset $\mathbb{M}$ of communication rules such that no other applicable rule can be added. Hence we have

$$\mathbb{M} \text{ is maximal } \Leftrightarrow \forall r \in \mathcal{R} \setminus \mathbb{M}, M + \mathbb{M}^- + M_r^- \ngeq 0 \tag{3}$$

where $\mathbb{M}^-$ is the multiset $\{\{M_r^{+ m_r} : r \in \mathbb{M} \wedge m_r = \text{mult}(r, \mathbb{M})\}\}$.

Given a configuration matrix M, finding a maximal set of rules is equivalent to finding non-negative integers, $m_k, 1 \leq k \leq |\mathcal{R}|$, such that the multiset $\{\{r_k^{m_k} : r_k \in \mathcal{R} \wedge m_k > 0\}\}$ is applicable and it cannot be extended by another applicable rule. In membrane computer literature there are many approaches to the way maximality is defined. As we are trying to apply membrane computing techniques to silicon computers, we are interested in a definition of maximality that fits well to this kind of devices. Therefore, we choose maximality in the sense of number of rules applied.

In this way, we find one of the possible maximal sets of rules from all of the available. For example, given $\Gamma = \{a, b\}$, the rules $r_1 = (1, a/b, 2)$ and $r_2 = (1, a/b^2)$ with multisets $w_1 = \{\{a^2\}\}$ and $w_2 = \{\{b^2\}\}$, the multisets $\mathbb{M}_1 = \{\{r_1^2\}\}$ and $\mathbb{M}_2 = \{\{r_2\}\}$ are both maximals. However, we prefer $\mathbb{M}_1$ because of the higher number of rule applications.

Hence our problem can be reduced to find non-negative integers m_k (thought to be multiplicities of rules) such that they maximize the sum $\sum_k m_k$ (which is

the total number of rule applications) subject to applicability condition. Therefore, this problem can be exposed as the following Integer Linear Programming problem

$$\begin{cases} \textbf{maximize:} \sum_{k=1}^{|\mathcal{R}|} m_k \\ \textbf{subject to:} \\ \sum_{k=1}^{|\mathcal{R}|} |M_k^-[i,j]| m_k \leq M[i,j], \text{ for } 0 \leq i \leq q, 1 \leq j \leq |\Gamma| \end{cases} \tag{4}$$

Although the Integer Linear Programming (ILP) problem in Equation 4 can be solved in parallel, it is a well known that it is a **NP** problem (with respect to the number of decision variables) [5, 8]. However, it can be solved with a reasonable speed in ordinary computers when the number of variables is relatively small. Hence it is important to decrease the number of decision variables. In order to do that we will divide the entire ILP problem in Equation 4 is several smaller problems. However, we must bear in mind the dependence between communication rules. Hence, we say that two communication rules $r = (i_u, u/v, i_v)$ and $r' = (i'_u, u'/v', i'_v)$ are *dependent* if they share some cells and, in the common ones, the multisets being communicated have non empty intersection. It is equivalent to the following condition

$$\begin{array}{c} i_u = i'_u \wedge u \cap u' \neq \emptyset \\ \vee \\ i_v = i'_v \wedge v \cap v' \neq \emptyset \\ \vee \\ i_u = i'_v \wedge u \cap v' \neq \emptyset \\ \vee \\ i_v = i'_u \wedge v \cap u' \neq \emptyset \end{array} \tag{5}$$

We can define a partition of the set of communication rules, $\mathcal{R}$, in several sets such that every two rules in distinct sets are independent. More formally, let $\sim$ be the relation in $\mathcal{R}$ given by $r \sim r'$ if and only if r and r' are dependent rules. Clearly $\sim$ is reflexive and symmetric. Let $\simeq$ be the transitive closure of $\sim$. With the definitions above, $\simeq$ is an equivalence relation and the quotient set $\mathcal{R}/\simeq$ defines a partition in the set of rules such that each rule in any set is independent with any rule in other partition set.

For each partition set defined above, we can define an ILP problem for rule selection as in Equation 4. Hence, we have a solution of the problem for the whole set of rules by considering the partial solution of each subproblem. This technique, together with an appropriate design of the rules, ensures an upper bound on the number of decision variables for each partial ILP making them available to be solved in reasonable time.

4 Example

In the following section, we will illustrate the techniques shown in section 3 (Encoding tissue-like P Systems using matrices) by the application of them to an example.

Let consider the following tissue-like P System with two cells

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

where

1. $\Gamma = \{a, b, c, d\}$ is the alphabet of objects;
2. $\Sigma = \emptyset$ is the input alphabet;
3. $\mathcal{E} = \{a, c, d\}$ represents the objects in the environment;
4. $w_1 = \{\{a^{10}, b^5, d\}\}$, $w_2 = \{\{a^4, c^7\}\}$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration;
5. $\mathcal{R}$ is the finite set of communication rules below:
   a) $r_1 = (0, c/ab, 1)$
   b) $r_2 = (0, c^5/a, 2)$
   c) $r_3 = (0, c^2/a^2b^3, 1)$
   d) $r_4 = (1, d/c^3, 2)$
   e) $r_5 = (0, a^2d^3/d, 1)$
6. $i_\Pi = 1$ is the input cell;
7. $o_\Pi = 2$ is the output cell.

Let M^k denote the configuration matrix of the k-th configuration of Π. Trivially,

$$M^0 = \begin{pmatrix} \infty & 0 & \infty & \infty \\ 10 & 5 & 0 & 1 \\ 4 & 0 & 7 & 0 \end{pmatrix}$$

If $\langle M_k^-, M_k^+ \rangle$ denotes the matricial form for rule r_k, for $k = 1, 2, 3, 4, 5$, then

$$M_1^- = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} M_1^+ = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} M_2^- = \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} M_2^+ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5 & 0 \end{pmatrix}$$

$$M_3^- = \begin{pmatrix} 0 & 0 & 2 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} M_3^+ = \begin{pmatrix} 2 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} M_4^- = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 \end{pmatrix} M_4^+ = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 5 & 1 \end{pmatrix}$$

$$M_5^- = \begin{pmatrix} 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} M_5^+ = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Rule dependency must be studied before ILP problem definition. Hence, the rule–dependency equivalence relation $\simeq$ decomposes $\mathcal{R}$ in two sets, being $[r_1] = \{r_1, r_2, r_3\}$ and $[r_4] = \{r_4, r_5\}$. Therefore, two ILP must be solved in order to find

a maximal set of rules to be applied for going from configuration i to $i+1$. These ILP will be respectively denoted as $ILP^{(1)}_{i\to\imath+1}$ and $ILP^{(4)}_{i\to\imath+1}$, and are represented below.

$$ILP^{(1)}_{i\to i+1}\begin{cases}\textbf{maximize:} m_1+m_2+m_3\\ \textbf{subject to:}\\ m_1+2m_3\leq \text{mult}(a,w_1)\\ m_1+3m_3\leq \text{mult}(b,w_1)\\ m_2\leq \text{mult}(a,w_2)\end{cases} \tag{6}$$

$$ILP^{(4)}_{i\to i+1}\begin{cases}\textbf{maximize:} m_4+m_5\\ \textbf{subject to:}\\ m_4+m_5\leq \text{mult}(d,w_1)\\ m_4\leq \text{mult}(c,w_2)\end{cases} \tag{7}$$

We firstly solve $ILP^{(1)}_{0\to 1}$, obtaining the solution $m_2=1$, $m_2=4$ and $m_3=1$. The problem $ILP^{(4)}_{0\to 1}$ has two maximal solutions, being $m_4=0, m_5=1$ and $m_4=1, m_5=0$. In such cases, we will choose one non deterministically. For example, let $m_4=0, m_5=1$ be the solution selected.

Both partial solutions make a maximal multiset $\mathbb{M}_{0\to 1}=\{\{r_1^2, r_2^4, r_3, r_5\}\}$, whose application arises the configuration given by matrix

$$M^1=\begin{pmatrix}\infty & 5 & \infty & \infty\\ 8 & 0 & 4 & 3\\ 0 & 0 & 35 & 0\end{pmatrix}$$

Analogously, at this step $ILP^{(1)}_{1\to 2}$ has only the solution $m_1=m_2=m_3=0$, while $ILP^{(4)}_{1\to 2}$ has multiple solutions given in the set $\{(0,3),(1,2),(2,1),(3,0)\}$, where the first one is m_4 and the last one is m_5. Again, one solution is non deterministically choosen, for example, $m_4=2, m_5=1$. Hence, the multiset of rules to be applied is $\mathbb{M}_{1\to 2}=\{\{r_4^2, r_5\}\}$, and the new configuration is given by matrix

$$M^2=\begin{pmatrix}\infty & 5 & \infty & \infty\\ 10 & 0 & 10 & 3\\ 0 & 0 & 29 & 2\end{pmatrix}$$

Next computation step involves solving $ILP^{(1)}_{2\to 3}$ and $ILP^{(4)}_{2\to 3}$. These Integer Linear Programs have the same solutions as previous ones. Again, in the second ILP one only solution must be non deterministically choosen. Let $m_4=2, m_5=1$ be that solution. Application of the multiset of rules $\mathbb{M}_{2\to 3}=\{\{r_4^2, r_5\}\}$ gives next configuration, settled as

$$M^3=\begin{pmatrix}\infty & 5 & \infty & \infty\\ 12 & 0 & 16 & 3\\ 0 & 0 & 23 & 4\end{pmatrix}$$

Next computation step is similar to the previous one and a solution from $\{(3,0),(2,1),(1,2),(0,3)\}$ must be non determistically choosen. Let it be $m_4 = 3, m_5 = 0$, for example. Therefore, the configuration matrix obtained is

$$M^4 = \begin{pmatrix} \infty & 5 & \infty & \infty \\ 12 & 0 & 25 & 0 \\ 0 & 0 & 14 & 7 \end{pmatrix}$$

On the next step, the only solution is the trivial one, settled as $m_k = 0, k = 1,2,3,4,5$, which is interpreted as halting condition.

5 Final Remarks

Parallelism is on the basis of Membrane Computing. All the theoretical efforts for designing membrane computing algorithms which use parallelism in an efficient way has the same problem in realistic simulations. Most of the current computers are sequential the simulations performed in such computers have the same bottleneck.

In this paper we report a preliminary work focused on a distributed simulation of tissue-like P systems in a cluster of computers. In a general view, if the number of cells does not increase along the computation it seems quite natural to plan a distribution of the work among several processors. The first steps in this line have led us to consider new representations (matrix representation) and new techniques (Integer Linear Programming) to solve the problems.

Many open research lines are open form this preliminary work. One of them is to consider different notions of parallelism [2], or introducing new features to our P system model in order to make it suitable for the hardware implementation.

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. Busi, N., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Efficient computation in rational-valued P systems. Mathematical Structures in Computer Science 19, 1125–1139 (2009), http://dx.doi.org/10.1017/S0960129509990144
2. Ciobanu, G., Marcus, S., Păun, Gh.: New strategies of using the rules of a P system in a maximal way. Power and complexity. Romanian Journal of Information Science and Technology (ROMJIST) 12, 157–173 (2009)

3. Ciobanu, G., Wenyuan, G.: P systems running on a cluster of computers. In: Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 2933, pp. 123–139. Springer (2003)
4. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Mario J. Pérez-Jiménez, M.: Software for P systems. In: Păun et al. [14], pp. 437–454.
5. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
6. Gershoni, R., Keinan, E., Păun, Gh., Piran, R., Ratner, T., Shoshani, S.: Research topics arising from the (planned) P systems implementation experiment in Technion. In: Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Sixth Brainstorming Week on Membrane Computing. pp. 183–192. Fénix Editora, Sevilla, Spain (2008)
7. 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)
8. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations. pp. 85 – 104. Plenum Press (1972)
9. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON. Lecture Notes in Computer Science, vol. 2387, pp. 290–299. Springer (2002)
10. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
11. Păun, Gh.: Computing with membranes. Tech. Rep. 208, Turku Centre for Computer Science, Turku, Finland (November 1998)
12. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000), see also [11]
13. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin, Germany (2002)
14. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
15. P system web page. http://ppage.psystems.eu

Linear Time Solution to Prime Factorization by Tissue P Systems with Cell Division

Xingyi Zhang[1], Yunyun Niu[2], Linqiang Pan[2], Mario J. Pérez-Jiménez[3]

[1] School of Computer Science and Technology
Anhui University, 230039 Hefei, China
xyzhanghust@gmail.com
[2] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology, 430074 Wuhan, China
niuyunyun1003@163.com, lqpan@mail.hust.edu.cn
[3] Department of Computer Science and Artificial Intelligence
University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Summary. Prime factorization is useful and crucial for public-key cryptography, and its application in public-key cryptography is possible only because prime factorization has been presumed to be difficult. A polynomial-time algorithm for prime factorization on a quantum computer is given by P. W. Shor in 1997. In this work, a linear-time solution for prime factorization is given on a kind of biochemical computational devices – tissue P systems with cell division, instead of physical computational devices.

1 Introduction

In math, *prime factorization* is the breaking down of a composite number into smaller primes, which when multiplied together equal the original integer. Currently, though the prime factorization problem is not known to be **NP**-hand, no efficient algorithm is publicly known. It is generally considered intractable. The presumed computational hardness of this problem is at the heart of several algorithms in cryptography such as RSA [15].

Many areas of mathematics and computer science have been brought to bear on the prime factorization problem, including elliptic curves, algebraic number theory, and quantum computing. A polynomial-time algorithm for prime factorization on a quantum computer is given by P. W. Shor in 1997 [16]. This will have significant implications for cryptography if a large quantum computer is ever built. However, before a practical quantum computer appears, it is still of interest to find any reasonable computational devices for solving prime factorization problem. In this work, we shall give a linear-time solution to prime factorization on a class of

biochemical computational devices – *tissue P systems with cell division*, instead of physical computational devices.

Tissue P systems with cell division is a class of computational devices in *membrane computing*. Membrane computing is an emergent branch of natural computing, which is inspired by the structure and the functioning of living cells, as well as the organization of cells in tissues, organs, and other higher order structures. The devices in membrane computing, called *P systems*, provide distributed parallel and non-deterministic computing models. Since Gh. Păun introduced the first P system in [12], this area is heavily investigated. Please refer to [13] for an introduction of membrane computing, and refer to [17] for further bibliography.

Informally, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous, non-deterministic, maximally parallel manner. *Tissue P systems* are a class of P systems, where membranes are placed in the nodes of a graph. It is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules, which was introduced to P systems in [11]. Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. This model has two biological inspirations (see [9]): intercellular communication and cooperation between neurons. In [14], tissue P systems are endowed with the ability of getting new cells based on the mitosis or cellular division, thus obtaining the ability of generating an exponential amount of workspace in polynomial time. Such variant of tissue P systems is called *tissue P systems with cell division*.

Tissue P systems with cell division were widely investigated for solving **NP**-complete problems. Some of them deal with non-numerical **NP**-complete decision problems, such as SAT problem [14], 3-coloring problem [2], vertex cover [4]. Others deal with numerical **NP**-complete decision problems, that is, decision problems whose instances consist of sets or sequences of integer numbers, such as subset sum [3], partition problem [5]. Although prime factorization we shall consider is a numerical problem, it is neither a decision problem nor an optimization problem. In this work, we shall construct a family of tissue P systems with cell division, which can decompose integer numbers in a linear time with respect to the length of binary representation of the integer to be factored. As a result of computation, a prime number is sent to a prefixed output membrane, instead of `yes` or `no`.

Up to now, besides there are two polynomial-time solutions to prime factorization by P systems with active membranes [6, 10], one well known polynomial algorithm that solves factorization problem is based on quantum computer [16]. As the case of quantum computer, the solution given in this work indicates how powerful tissue P systems with cell division can be, although at this moment nobody knows how to build a biochemical computer.

The paper is organized as follows. In Section 2, some preliminaries are recalled. The formal definition of tissue P systems with cell division is given in Section 3. A family of tissue P systems that uniformly solve the factorization problem is

presented in Section 4, with a short overview of the computation and the necessary resources. Conclusions and comments are presented in Section 5.

2 Preliminaries

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

A *multiset* m over a set A is a pair (A, f), where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset, then its *support* is defined as $supp(m) = \{x \in A \,|\, f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over A, and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{\{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}\}$. That is, superscripts indicate the multiplicity of each element. If $f(x) = 0$ for any $x \in A$, then this element is omitted.

3 Tissue P Systems with Cell Division

In [8, 9], the first definition of the model of tissue P systems was proposed, where the membrane structure did not change along the computation. We now shall introduce a model of *tissue P systems with cell division* based on the cell-like model of P systems with membranes division [14]. The biological inspiration of this model is clear: alive tissues are not *static* network of cells, since new cells are generated by membrane fission in a natural way.

The main features of this model, from the computational point of view, are that cells are not polarized (the contrary holds in the cell-like model of P systems with active membranes, see [13]); the cells obtained by division have the same labels as the original cell and if a cell is divided, its interaction with other cells or with the environment is blocked during the division process. In some sense, this means that while a cell is dividing it closes its communication channels with other cells and with the environment.

Formally, a *(function) computing tissue P system with cell division* of degree $q \geq 1$ and order (m, n), $m \geq 1, n \geq 1$, is a tuple of the form

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

where:

1. Γ is the *alphabet* of *objects*;

2. $\Sigma = \{a_1, \ldots, a_m\}$ is an ordered input alphabet strictly contained in Γ;
3. $\Lambda = \{b_1, \ldots, b_n\}$ is an ordered output alphabet contained in Γ;
4. $w_1, \ldots, w_q$ are strings over Γ, describing the initial multisets of objects placed in the cells of the system at the beginning of the computation;
5. $\mathcal{E} \subseteq \Gamma$ is the set of objects in the environment in arbitrarily copies each;
6. $\mathcal{R}$ is a finite set of rules of the following forms:
   (a) $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;
   Communication rules; $1, 2, \cdots, q$ identify the cells of the system, 0 is the environment; when applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and simultaneously the objects of the multiset v are sent from region j to region i ($|u| + |v|$ is called the length of the communication rule $(i, u/v, j)$);
   (b) $[a]_i \rightarrow [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$, $a, b, c \in \Gamma$, and $i \neq i_{out}$;
   Division rules; in reaction with an object a, the cell is divided into two cells with the same label; all the objects in the original cells are replicated and copies of them are placed in each of the new cells, with the exception of the object a, which is replaced by the object b in the first new cell and by c in the second one; the output cell cannot be divided;
7. $i_{in} \in \{1, 2, \ldots, q\}$ is the input cell;
8. $i_{out} \in \{0, 1, 2, \ldots, q\}$ is the output cell.

The rules of a system as above are used in the non-deterministic maximally parallel manner. In each step, all cells which can evolve must evolve in a maximally parallel way (in each step we apply a multiset of rules which is maximal, no further rule can be added). This way of applying rules has only one restriction when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell do not evolve by means of communication rules. Their labels precisely identify the rules which can be applied to them.

A configuration of tissue P system with cell division is described by all multisets of objects over Γ associated with all the cells present in the system and the multiset of objects over $\Gamma - \mathcal{E}$ associated with environment. The initial configuration of the system Π with input $w \in \Sigma^*$ is the tuple $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \emptyset)$; that is, the corresponding configuration after adding the multiset w to the content of the input cell i_{in}. The computation starts from the initial configuration and proceeds as defined above. When there is no rule can be applied, the computation stops. Only halting computations give a result. If $\mathcal{C} = \{C^i\}_{i<r}$ is a halting computation, where C^i are configurations, then the result of computation $Output(\mathcal{C}) = (C^{r-1}_{b_1}(i_{out}), C^{r-1}_{b_2}(i_{out}), \ldots, C^{r-1}_{b_n}(i_{out}))$, where $C^{r-1}_{b_j}(i_{out})$, $1 \leq j \leq n$, is the multiplicity of object b_j in the region i_{out} in the halting configuration C^{r-1}.

For a function f, we denote the domain of f by $D(f)$ and the range of f by $R(f)$. For a tissue P system with cell division Π having ordered input alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$ and ordered output alphabet $\Lambda = \{b_1, b_2, \ldots, b_n\}$, and partial function $f : \mathbb{N}^m \rightarrow \mathbb{N}^n$, function f is encoded in a unary notation in the following

way: $(\alpha_1, \ldots, \alpha_m) \in D(f)$ is expressed by $a_1^{\alpha_1} a_2^{\alpha_2} \ldots a_m^{\alpha_m}$; $(\beta_1, \ldots, \beta_n) \in R(f)$ is expressed by $b_1^{\beta_1} b_2^{\beta_2} \ldots b_n^{\beta_n}$.

Definition 1. *We say that a partial function $f : \mathbb{N}^m \to \mathbb{N}^n$ is computed in polynomial time by a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of tissue P systems with cell division in unary encoding if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(t)$ from $t \in \mathbb{N}$.*
- *There exist a polynomial-time computable function s over the domain $D(f)$ of function f such that:*
  - *for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$, $s(u)$ is a natural number and $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ is an input multiset of the system $\Pi(s(u))$;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (f, s), that is, there exists a polynomial function p, such that for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$ every computation of $\Pi(s(u))$ with input $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ is halting and, moreover, it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (f, s), that is, for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$, if there exists a computation $\mathcal{C}$ of $\Pi(s(u))$ with input $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ such that $Output(\mathcal{C}) = (\beta_1, \ldots, \beta_n)$, then $f(u) = (\beta_1, \cdots, \beta_n)$;*
  - *the family $\mathbf{\Pi}$ is* complete *with regard to (f, s), that is, for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$, if $f(u) = (\beta_1, \cdots, \beta_n)$, then every computation $\mathcal{C}$ of $\Pi(s(u))$ with input $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ has $Output(\mathcal{C}) = \{\beta_1, \ldots, \beta_n\}$.*

In the Definition 1, the input and output are encoded in unary notation. However, in classical complexity theory, based upon Turing machine, switching from binary to unary encoding generally corresponds to simplify the problem. In this work, binary encoding is used for integer factorization problem. In what follows, we will give the definition that a function is computed by a family of P systems with cell division in binary encoding. In the case of binary encoding, the input alphabet is not asked to be ordered, and no output alphabet is fixed.

A *(function) computing tissue P system with cell division with input* of degree $q \geq 1$ is a tuple of the form

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

where:

1. Γ is the *alphabet* of *objects*;
2. Σ is an (un-ordered) input alphabet strictly contained in Γ;
3. $w_1, \ldots, w_q$ are strings over Γ, describing the initial multisets of objects placed in the cells of the system at the beginning of the computation;
4. $\mathcal{E} \subseteq \Gamma$ is the set of objects in the environment in arbitrarily copies each;
5. $\mathcal{R}$ is a finite set of rules of the following forms:
   (a) $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;

(b) $[a]_i \to [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$, $a, b, c \in \Gamma$, and $i \neq i_{out}$;
6. $i_{in} \in \{1, 2, \ldots, q\}$ is the input cell;
7. $i_{out} \in \{0, 1, 2, \ldots, q\}$ is the output cell.

In semantics, P systems having un-ordered alphabets is the same with P systems with ordered input and output alphabets except for the way of encoding input and output. In the unary encoding, the sizes of ordered input and output alphabets are related with the dimensions of domain and range of function that is computed. Specifically, an ordered input alphabet $\{a_1, \ldots, a_m\}$ and an ordered output alphabet $\{b_1, \ldots, b_n\}$ can encode each function whose domain (resp. range) is a subset of $\mathbb{N}^{m'}$ $(m' \leq m)$ (resp. $\mathbb{N}^{n'}$ $(n' \leq n)$). In the binary encoding, the size of alphabet is related with both input value and output value. For example, for the function $f(x) = 2^{2^x}$ $(x \in \mathbb{N})$ and an input n, the length of input n in binary expression is $\lfloor \lg n \rfloor + 1$, and the length of output $f(n)$ in binary expressions is $2^n + 1$, which is an exponential function with respect to $\lfloor \lg n \rfloor + 1$. For functions such as $f(x) = 2^{2^x}$, maybe, we need exponential (with respect to the input size) large alphabet to encode the function in P systems, hence we cannot construct a family of P systems with cell division in polynomial time by Turing machine to compute functions such as $f(x) = 2^{2^x}$. It depends on the property of function whether a function can be computed by tissue P systems with cell division in binary encoding.

For prime factorization problem, the factors are less than the integer to be factored. In fact enables us to find a reasonable binary encoding for prime factorization problem. Specifically, we shall use the method from [7] to encode binary numbers by multisets of objects. Let $x_{k-1}, \cdots, x_1, x_0$ (with $k \geq 1$) be the binary representation of integer $x \geq 0$, that is, $x = \sum_{i=0}^{k-1} x_i 2^i$. We use the objects from the following alphabet $\mathcal{A}_k$, for $k \geq 1$:

$$A_k = \{\langle b, j \rangle \mid b \in \{0, 1\}, j \in \{1, 2, \cdots, k\}\}.$$

Objects $\langle b, j \rangle$ is used to represent bit b in position j in the binary encoding of an integer number. Hence, to represent the above number x we will use the following multiset (actually, a set) of objects:

$$\langle x_{k-1}, k-1 \rangle, \cdots, \langle x_1, 1 \rangle, \langle x_0, 0 \rangle.$$

Let us remark that the alphabet $\mathcal{A}_k$ depends on the length of the binary representation of the number x. Moreover, it is clear that with $\mathcal{A}_k$ we can represent all integer numbers in the range $0, 1, \cdots, 2^k - 1$. In order to distinguish between the objects that represent the bits of different integers A and B, a leading label A, B are used to mark each element in the multiset. To this aim, the alphabet $\mathcal{A}_k$ is modified as follows:

$$\mathcal{A}'_k = \{\langle l, b, j \rangle \mid l \in \{A, B\}, b \in \{0, 1\}, j \in \{1, 2, \cdots, k\}\}.$$

In this way, the i-th bit of A (that is, a_i) and the j-th bit of B (that is, b_j) are represented by the objects $\langle A, a_i, i \rangle$ and $\langle B, b_j, j \rangle$, respectively.

In general, we give the following definition that a function is computed by P systems with cell division in binary encoding.

Definition 2. *We say that a partial function $f : \mathbb{N} \to \mathbb{N}$ is computed in polynomial time by a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of tissue P systems with cell division in binary encoding if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(t)$ from $t \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions over the domain $D(f)$ of function f such that:*
  - *for each $u \in D(f)$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (f, cod, s), that is, there exists a polynomial function p, such that for each $u \in D(f)$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (f, cod, s), that is, for each $u \in D(f)$, if there exists a computation $\mathcal{C}$ of $\Pi(s(u))$ with input $cod(u)$ and the objects in region i_{out} in the last configuration of $\mathcal{C}$ encode $(\beta_1, \cdots, \beta_q) \in \mathbb{N}^q$, then $f(u) = (\beta_1, \cdots, \beta_q)$;*
  - *the family $\mathbf{\Pi}$ is* complete *with regard to (f, cod, s), that is, for each $u \in D(f)$, if $f(u) = (\beta_1, \cdots, \beta_q) \in \mathbb{N}^q$, then in every computation of $\Pi(s(u))$ with input $cod(u)$, the objects in region i_{out} in the last configuration encode $(\beta_1, \cdots, \beta_q)$.*

4 A Linear Time Solution to the Factorization Problem

When we discuss the prime factorization problem, it is necessary to distinguish two different versions of the problem: decision problem version and function problem version.

The decision problem version of prime factorization can be formulated as "is n a composite number?" (or equivalently: "is n a prime number?"). This version is natural and useful because most well-studied complexity classes are defined as classes of decision problems, not function problems. But the decision problem version of prime factorization is much easier than the problem of finding the factors of n. Specifically, it can be solved in polynomial time (with respect to the number of digits of n) with the AKS primality test [1].

The function problem version of prime factorization: given an integer n, find an integer d with $1 < d < n$ that divides n (or conclude that n is prime). It is trivially in the class FNP, but we do not known whether it lies in class FP or not. This version is generally considered intractable, which means that no polynomial-time (with respect to the instance size) algorithm is known that solves it on every

instance; and it is the version solved by most practical implementations. In this work, we shall consider a restricted version of prime factorization problem, based on the following two facts. (1) Given an algorithm for integer factorization, one can factor any integer down to its constituent prime factors by repeated application of this algorithm. (2) Not all numbers of a given length are equally hard to factor. Semiprimes (the product of two prime numbers) are believed as the hardest instances of integer factorization for currently known techniques.

Problem 1. NAME: factorization.
– INSTANCE: a positive integer number which is the product of two prime numbers.
– OUTPUT: the prime factor that is not greater than another one.

Next, we shall construct a family $\{\Pi(k)\}_{k\in\mathbb{N}}$ of tissue P systems with cell division to factor integers, where each system $\Pi(k)$ can decompose all numbers of length k in binary form, provided that an appropriate input multiset is given. The resolution is a brute force algorithm, which consists of the following stages:

- *Generation Stage*: By division, all the possible pairs of integer numbers of length k in binary form are produced (one pair for each membrane with label 2).
- *Pre-checking Stage*: In this stage, the product of each pair of integer numbers of length k is calculated.
- *Checking Stage*: The system checks whether or not there exists a pair of integer numbers such that their product equals to the number n to be composed.
- *Output Stage*: The system sends to the output region a prime number.

For each $k \in \mathbb{N}$,

$$\Pi(k) = (\Gamma(k), \Sigma(k), w_1, w_2, \mathcal{R}(k), \mathcal{E}(k), i_{in}, i_{out}),$$

with the following components:

- $\Gamma(k) = \Sigma(k) \cup \{a_i, b_i, \langle X, 0, i\rangle, f_i, g_i \mid 0 \le i \le k-1\} \cup$
  $\{\langle A, j, i\rangle, \langle B, j, i\rangle, \langle A', j, i\rangle, \langle B', j, i\rangle \mid 0 \le i \le k-1, 0 \le j \le 1\} \cup$
  $\{\langle A_j, l, i\rangle, \langle B_j, l, i\rangle \mid 0 \le i \le k-1, 0 \le j \le \lceil \lg k\rceil + 1, 0 \le l \le 1\} \cup$
  $\{c_i \mid 0 \le i \le 4k + \lceil \lg 2k\rceil + \lceil \lg k\rceil + 5\} \cup \{c'_i \mid 1 \le i \le \lceil \lg k\rceil + 2k + 3\} \cup$
  $\{\langle C, 0, i\rangle, \langle C, 1, i\rangle \mid 0 \le i \le 2k-1\} \cup \{\langle i, j\rangle \mid 0 \le i, j \le k-1\} \cup$
  $\{d_i \mid -1 \le i \le k-2\} \cup \{e_i \mid -1 \le i \le k-1\} \cup \{z\}$.
- $\Sigma(k) = \{\langle n, 0, i\rangle, \langle n, 1, i\rangle \mid 0 \le i \le k-1\}$.
- $w_1 = \{\{c_0\}\}$.
- $w_2 = \{\{a_0 a_1 \cdots a_{k-1} b_0 b_1 \cdots b_{k-1} z\}\} \cup \{\{\langle i, j\rangle \mid 0 \le i, j \le k-1\}\}$.
- $\mathcal{R}(k)$ is the set of rules:

  1. **Division rule:**
     $r_{1,i} \equiv [a_i]_2 \to [\langle A, 0, i\rangle]_2[\langle A, 1, i\rangle]_2$, for $0 \le i \le k-1$;
     $r_{2,i} \equiv [b_i]_2 \to [\langle B, 0, i\rangle]_2[\langle B, 1, i\rangle]_2$, for $0 \le i \le k-1$.

2. **Communication rules:**

$r_{3,i} \equiv (1, c_i/c_{i+1}^2, 0)$, for $0 \leq i \leq 2k-1$;
$r_4 \equiv (1, c_{2k}/z, 2)$;
$r_{5,i} \equiv (2, c_{2k+i}/c_{2k+i+1}^2, 0)$, for $0 \leq i \leq \lceil \lg 2k \rceil - 1$;
$r_{6,i,j} \equiv (2, c_{2k+\lceil \lg 2k \rceil}\langle A, j, i\rangle / c_{2k+\lceil \lg 2k \rceil+1}\langle A_0, j, i\rangle, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{7,i,j} \equiv (2, c_{2k+\lceil \lg 2k \rceil}\langle B, j, i\rangle / c_{2k+\lceil \lg 2k \rceil+1}\langle B_0, j, i\rangle, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_8 \equiv (2, c_{2k+\lceil \lg 2k \rceil+1}/c'_1 c_{2k+\lceil \lg 2k \rceil+2}, 0)$;
$r_{9,i} \equiv (2, c_{2k+\lceil \lg 2k \rceil+i}/c_{2k+\lceil \lg 2k \rceil+i+1}, 0)$, for $2 \leq i \leq \lceil \lg k \rceil + 2k + 4$;
$r_{10,i} \equiv (2, c'_i/c'_{i+1}, 0)$, for $1 \leq i \leq \lceil \lg k \rceil + 2k + 2$;
$r_{11,i,j,l} \equiv (2, \langle A_j, l, i\rangle / \langle A_{j+1}, l, i\rangle^2, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq \lceil \lg k \rceil$, $0 \leq l \leq 1$;
$r_{12,i,j,l} \equiv (2, \langle B_j, l, i\rangle / \langle B_{j+1}, l, i\rangle^2, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq \lceil \lg k \rceil$, $0 \leq l \leq 1$;
$r_{13,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle \langle B_{\lceil \lg k \rceil+1}, 0, j\rangle \langle i, j\rangle / \langle C, 0, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{14,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle \langle B_{\lceil \lg k \rceil+1}, 1, j\rangle \langle i, j\rangle / \langle C, 0, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{15,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle \langle B_{\lceil \lg k \rceil+1}, 0, j\rangle \langle i, j\rangle / \langle C, 0, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{16,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle \langle B_{\lceil \lg k \rceil+1}, 1, j\rangle \langle i, j\rangle / \langle C, 1, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{17,i} \equiv (2, \langle C, 0, i\rangle \langle C, 0, i\rangle / \langle C, 0, i\rangle, 0)$, for $0 \leq i \leq 2k-2$;
$r_{18,i} \equiv (2, \langle C, 0, i\rangle \langle C, 1, i\rangle / \langle C, 1, i\rangle, 0)$, for $0 \leq i \leq 2k-2$;
$r_{19,i} \equiv (2, \langle C, 1, i\rangle \langle C, 1, i\rangle / \langle C, 0, i\rangle \langle C, 1, i+1\rangle, 0)$, for $0 \leq i \leq 2k-2$;
$r_{20,i,j} \equiv (2, c'_{\lceil \lg k \rceil+2k+3}\langle C, 1, i\rangle \langle n, j, k-1\rangle / \lambda, 0)$,
for $k \leq i \leq 2k-2$, $0 \leq j \leq 1$;
$r_{21,i,j} \equiv (2, c_{4k+\lceil \lg 2k \rceil+\lceil \lg k \rceil+5}\langle C, j, i\rangle \langle n, j, i\rangle / \langle X, 0, i\rangle, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{22} \equiv (2, \langle X, 0, k-1\rangle / d_{k-2}, 0)$;
$r_{23,i} \equiv (2, d_i\langle X, 0, i\rangle / d_{i-1}, 0)$, for $0 \leq i \leq k-2$;
$r_{24} \equiv (2, d_{-1}/e_{k-1}, 0)$;
$r_{25,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, j, i\rangle \langle B_{\lceil \lg k \rceil+1}, j, i\rangle e_i / \langle A_{\lceil \lg k \rceil+1}, j, i\rangle$
$\langle B_{\lceil \lg k \rceil+1}, j, i\rangle e_{i-1}, 0)$, for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{26} \equiv (2, e_{-1}/f_0, 0)$;
$r_{27,i} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle \langle B_{\lceil \lg k \rceil+1}, 0, i\rangle e_i / \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle$
$\langle B_{\lceil \lg k \rceil+1}, 0, i\rangle f_0, 0)$, for $0 \leq i \leq k-1$;
$r_{28,i,j} \equiv (2, f_i\langle B_{\lceil \lg k \rceil+1}, j, i\rangle / f_{i+1}\langle B', j, i\rangle, 0)$, for $0 \leq i \leq k-2$, $0 \leq j \leq 1$;
$r_{29,j} \equiv (2, f_{k-1}\langle B_{\lceil \lg k \rceil+1}, j, k-1\rangle / \langle B', j, k-1\rangle, 0)$, for $0 \leq j \leq 1$;
$r_{30,i,j} \equiv (2, \langle B', j, i\rangle / \lambda, 3)$, for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{31,i} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle \langle B_{\lceil \lg k \rceil+1}, 1, i\rangle e_i / \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle$
$\langle B_{\lceil \lg k \rceil+1}, 1, i\rangle g_0, 0)$, for $0 \leq i \leq k-1$;
$r_{32,i,j} \equiv (2, g_i\langle A_{\lceil \lg k \rceil+1}, j, i\rangle / g_{i+1}\langle A', j, i\rangle, 0)$, for $0 \leq i \leq k-2$, $0 \leq j \leq 1$;

$r_{33,j} \equiv (2, g_{k-1}\langle A_{\lceil \lg k \rceil+1}, j, k-1\rangle / \langle A', j, k-1\rangle, 0)$, for $0 \leq j \leq 1$;
$r_{34,i,j} \equiv (2, \langle A', j, i\rangle / \lambda, 3)$, for $0 \leq i \leq k-1$, $0 \leq j \leq 1$.

- $\mathcal{E}(k) = \Gamma(k)$.
- $i_{in} = 2$ is the *input cell*.
- $i_{out} = 3$ is the *output cell*.

4.1 An Overview of the Computation

A family of tissue P systems with cell division is constructed as above. Let n be an instance of the prime factorization problem, where n is the integer number to be decomposed and k is the total number of binary bits to represent n. Then we consider a size mapping on the set of instances defined as $s(u) = k$. The coding of the instance is the multiset $cod(u) = \langle n, i_{k-1}, k-1\rangle\langle n, i_{k-2}, k-2\rangle \cdots \langle n, i_0, 0\rangle$, where $i_j = 0$ or 1 $(0 \leq j \leq k-1)$ is the bit at position j in the binary encoding of n. In what follows, we will informally describe how the tissue P system with cell division $\Pi(s(u))$ with input $cod(u)$ works.

Let us start with the generation stage. This stage has two parallel processes, which is described in two items.

- On one hand, in the cell with label 1 by using the rule $r_{3,i}$ the object c_i is multiplied until step $2k$; starting from c_0 object c_i grows its subscript by one in each step. Therefore, 4^k copies of c_{2k} are obtained in the cell with label 1 at step $2k$.
- On the other hand, in the cell with label 2 the division rules $r_{1,i}$ and $r_{2,i}$ are applied. For each object a_i (which is used to generate the two possible bits at position i in the binary encoding of integer number A), two cells labeled by 2 are produced, one of them containing a new object $\langle A, 0, i\rangle$ and the other one containing another new object $\langle A, 1, i\rangle$. Object $\langle A, 0, i\rangle$ (resp. $\langle A, 1, i\rangle$) represents the fact that the bit at position i in the binary encoding of A is 0 (resp. 1). Similarly, for each object b_i (which is used to generate the two possible bits at position i in the binary encoding of integer number B), two cells labeled by 2 are also produced, one of them containing a new object $\langle B, 0, i\rangle$ and the other one containing another new object $\langle B, 1, i\rangle$. The objects a_i, b_i are non-deterministically chosen, after $2k$ steps of division we obtain exactly 4^k cells with label 2, each of them encoding one possible pair of integer numbers A and B whose values range from 0 to $2^k - 1$. The object z is duplicated, hence a copy of z appears in each cell with label 2. Note that after step $2k$ the cells with label 2 cannot divide any more, because the objects a_i and b_i are exhausted.

The pre-checking stage starts from step $2k + 1$, in this stage, the product of each pair of integer numbers in cell with label 2 is calculated. At the step $2k + 1$, there are 4^k copies of c_{2k} in the cell with label 1, and there are 4^k cells with label 2, each of them containing a copy of z, so the rule r_4 is enabled and applied.

Due to the maximality of the parallelism of using the rule r_4, each cell with label 2 gets precisely one copy of c_{2k}. In the next $\lceil \lg 2k \rceil$ steps, by the rule $r_{5,i}$, the object c_{2k+i} is duplicated and its subscript increases by one in each step; so at step $2k + \lceil \lg 2k \rceil + 1$, there are at least $2k$ copies of $c_{2k+\lceil \lg 2k \rceil}$ in each cell with label 2. Once object $c_{2k+\lceil \lg 2k \rceil}$ is generated, by the rules $r_{6,i,j} - r_{7,i,j}$, each copy of objects $\langle A, j, i \rangle$ and $\langle B, j, i \rangle$, $0 \leq i \leq k-1$, $0 \leq j \leq 1$, together with a copy of object $c_{2k+\lceil \lg 2k \rceil}$, is traded for one copy of objects $\langle A_0, j, i \rangle$, $\langle B_0, j, i \rangle$, and one copy of object $c_{2k+\lceil \lg 2k \rceil+1}$ at step $2k + \lceil \lg 2k \rceil + 2$.

From step $2k + \lceil \lg 2k \rceil + 3$ to step $2k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 3$, by the rules $r_{11,i,j,l}$ and $r_{12,i,j,l}$, the objects $\langle A_j, l, i \rangle$ and $\langle B_j, l, i \rangle$ duplicate themselves until getting at least $k+1$ copies of objects $\langle A_{\lceil \lg k \rceil+1}, l, i \rangle$ and $\langle B_{\lceil \lg k \rceil+1}, l, i \rangle$. In the following computation, k copies of these objects are used to obtain the product of integer numbers A and B, the other one copy of these objects is used to output the computing result.

For any two k-bits integer numbers $A = \sum_{i=0}^{k-1} x_i 2^i$ ($x_i = 0$ or 1) and $B = \sum_{i=0}^{k-1} y_i 2^i$ ($y_i = 0$ or 1), the product of A and B can be written as $A \times B = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} x_i y_j 2^{i+j}$. In order to get the product, we will first compute the contribution of each pair of bits x_i and y_j, and then sum all the contributions. Specifically, the rules $r_{13,i,j} - r_{16,i,j}$ are used to get the contribution of each pair of bits; the rules $r_{17,i} - r_{19,i}$ are used to get the sum of all contributions. Since each cell with label 2 contains at least $k+1$ copies of objects $\langle A_{\lceil \lg k \rceil+1}, j, i \rangle$ and $\langle B_{\lceil \lg k \rceil+1}, j, i \rangle$, $j = 1$ or 0, the process of computing the product of each pair of bits a_i and b_j only needs one step, which produces some bits of the result C as well as the carry bits. It takes at most $2k$ steps to sum all the bits by the rules $r_{17,i} - r_{19,i}$. In this way, after step $4k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 4$ the product of two integer numbers in each cell with label 2 is computed and the pre-checking stage is finished. At this moment, the subscript of object c_j reaches $4k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 4$ by the rules r_8 and $r_{9,i}$, while the subscript of object c'_i reaches $\lceil \lg k \rceil + 2k + 3$ by the rule $r_{10,i}$.

The checking stage starts from step $4k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 5$ with the application of the rules $r_{20,i,j}$, $r_{21,i,j}$, r_{22} and $r_{23,i}$. The rules $r_{20,i,j}$ are used to check whether the "efficient" length of bits of the product is greater than $k-1$, thus whether there exists at least one position i, $k \leq i \leq 2k-2$, in the product having bit 1. If there exists such position in the binary encoding of the product, then the product must be greater than n. In this case, at least one of objects $\langle C, 1, i \rangle$, $k \leq i \leq 2k-2$, must appear in this cell. By the rule $r_{20,i,j}$, objects $\langle n, j, k-1 \rangle$, $j = 0$ or 1, are removed. The rules $r_{21,j,k-1}$ and r_{20} cannot be used without object $\langle n, j, k-1 \rangle$, thus this cell with label 2 will not send objects to the output cell. If the bit on each position i such that $k \leq i \leq 2k-2$ equals to 0, then at that step only the rule can be used by which object $c_{4k+\lceil \lg 2k \rceil+\lceil \lg k \rceil+4}$ is traded for $c_{4k+\lceil \lg 2k \rceil+\lceil \lg k \rceil+5}$. The rues $r_{21,i,j}$, r_{22} and $r_{23,i}$ are used to check whether the product equals to n.

- If *the product equals to* n in a cell with label 2, then all objects $\langle X, 0, i \rangle$, $0 \leq i \leq k-1$, should be produced in this cell by the rule $r_{21,i,j}$. The rules r_{22} and $r_{23,i}$ are used to check whether all objects $\langle X, 0, i \rangle$, $0 \leq i \leq k-1$, are

produced. It is performed one bit by one bit, starting from the most significant bit. The object $\langle X, 0, k-1\rangle$ is traded for d_{k-2}, then d_{k-2} and $\langle X, 0, k-2\rangle$ are traded for d_{k-3}, the process continues until the position 0 is checked and the object d_{-1} is produced. Since the length of the integer number n is k, this process takes k steps. The computation passes to the output stage from step $5k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 7$.

- If *the product does not equal to* n in a cell with label 2, then at least one object $\langle X, 0, i\rangle$, $0 \leq i \leq k-1$, does not be produced in this cell. Without loss of generality, we assume that the first object without appearing in this cell is $\langle X, 0, s\rangle$, starting from the most significant bit, where $0 \leq s \leq k-1$. By the rules r_{22} and $r_{23,i}$, it is not difficult to find that object d_{s-1} will not be produced and the computation of the system halts at that moment. That means that this cell will not send any objects to the output cell.

The output stage starts from step $5k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 7$. In this stage, if the product of two integer numbers equals to n and the two numbers are also equal, then one of them will be outputted to the output cell; if the product of two integer numbers equals to n and the two numbers are not equal, then the smaller one will be outputted to the output cell. According to the checking stage, if the product of two integer numbers equals to n in a cell with label 2, then the object d_{-1} appears in this cell. The object d_{-1} is traded for e_{k-1}. The rules $r_{25,i,j}$, r_{26} and $r_{31,i}$ are used to check which integer number is smaller or whether they are equal. If object e_{-1} appears, then it means the fact that two integer numbers are equal, and the integer number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$ is outputted to the output cell labeled by 3 by the rules r_{26}, $r_{28,i,j}$, $r_{29,j}$, and $r_{30,i,j}$. If two integer numbers are not equal, then object f_0 or g_0 should appear and object e_{-1} does not appear. If object f_0 appears in the cell with label 2, it means the fact that the integer number corresponding to objects $\langle A_{\lceil \lg k \rceil+1}, j, i\rangle$ is greater than the integer number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$, and the integer number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$ is outputted to the output cell with label 3 by the rules r_{26}, $r_{28,i,j}$, $r_{29,j}$, and $r_{30,i,j}$. If object g_0 appears in the cell with label 2, it means the fact that the integer number corresponding to objects $\langle A_{\lceil \lg k \rceil+1}, j, i\rangle$ is less than the number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$, and the integer number corresponding to objects $\langle A_{\lceil \lg k \rceil+1}, j, i\rangle$ is outputted to the output cell with label 3 by the rules $r_{32,i,j}$ $r_{33,j}$, and $r_{34,i,j}$. This stage takes not more than $2k+3$ steps, and in the case that two integer numbers are equal, the output stage takes exactly $2k + 3$ steps. So the computation of the system stops after step $7k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 9$, and the computation result can read out from the objects in the output cell with label 3.

4.2 A Simple Example

In order to show how the tissue P systems with cell division constructed in Section 4.1 work, let us consider the factorization of integer number 2. Hence, we have $n = 2$ and $k = 2$. The initial configuration of tissue P system with cell division

$\Pi(2)$ for the factorization of integer number 2 is illustrated in Figure 1. From the figure, it can be found that 2 is represented by objects $\langle n, 1, 1\rangle$ and $\langle n, 0, 0\rangle$.

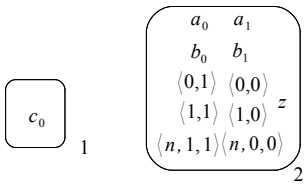


Fig. 1. The initial configuration of system $\Pi(2)$ for factoring integer number 2

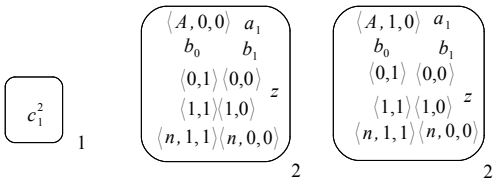


Fig. 2. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 1

At step 1, both the cell with label 1 and the cell with label 2 have rules which can be used. In the cell with label 1, by the rule $r_{3,0}$ object c_0 evolves to c_1 and its number is doubled; in the cell with label 2, objects a_0, a_1, b_0 and b_1 are non-deterministically chosen to divide this cell. Without loss of generality, we assume that a_0 is used to divide the cell with label 2. Object a_0 is consumed and two objects $\langle A, 0, 0\rangle$ and $\langle A, 1, 0\rangle$ are generated, with one object appearing in a cell with label 2 and another one appearing in the other cell with label 2. The other objects in the cell with label 2 are duplicated in the two new cell with label 2. The configuration of the system at this step is shown in Figure 2.

Similar to the work of object a_0, objects a_1, b_0 and b_1 can continue to divide the cells with label 2 in the following three steps, with one object dividing its corresponding cell one time. At the same time, in cell with label 1 the number of object c_4 becomes 16. The configuration of the system at step 4 is shown in Figure 3. Note that at this moment all pairs of integer numbers of length 2 are generated, with one cell with label 2 containing a pair.

After step 4, the system enters to the pre-checking stage. In this stage, the product of each pair of integer numbers is calculated. This process is done in parallel in the cells with label 2. The product of each pair of integer numbers is represented by objects $\langle C, i, j\rangle$, $i = 0$ or 1, $0 \leq j \leq 3$. Figure 4 gives the configuration of the system when the pre-checking stage is finished. The pre-checking stage finishes at step 15.

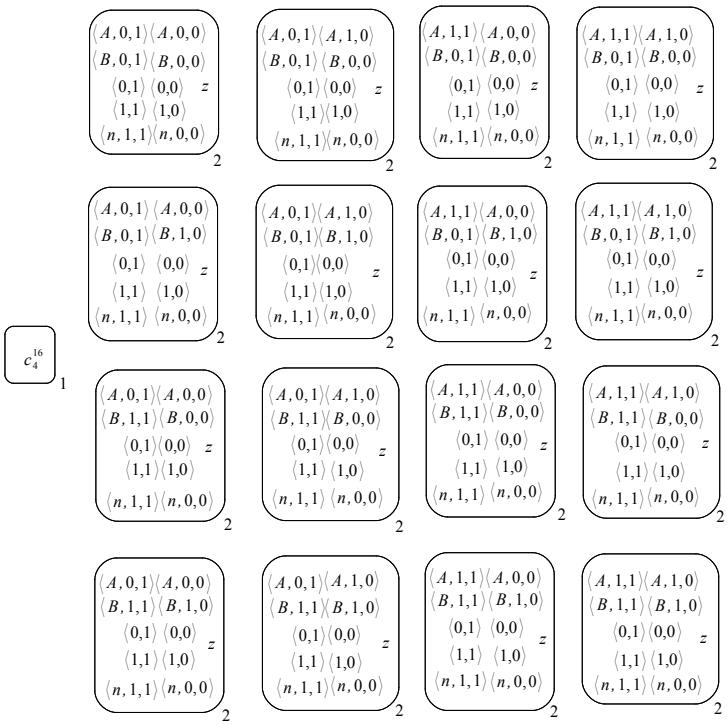


Fig. 3. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 4

The checking stage starts at step 16. In this stage, the system compares each product represented by objects $\langle C, i, j\rangle$, $i = 0$ or 1, $0 \leq j \leq 3$, with the integer number 2 represented by objects $\langle n, 1, 1\rangle$ and $\langle n, 0, 0\rangle$. If there exists at least one position i, $2 \leq i \leq 3$, at which the bit of a product equals to 1 (that is, there exists object $\langle C, 1, 2\rangle$ or $\langle C, 1, 2\rangle$), then this object together with objects $\langle n, 1, 1\rangle$ and c'_8 is removed from the corresponding cell with label 2. If the product equals to the integer number 2, then both object $\langle X, 0, 1\rangle$ and object $\langle X, 0, 0\rangle$ will appear in the corresponding cell with label 2. Figure 5 gives the configuration of the system when the checking stage is finished.

The output stage starts at step 20. In this stage, each cell with label 2 which contains the product that equals to the integer number 2, outputs the integer number that is not greater than another one. Such integer number is represented by objects of the form $\langle A', i, j\rangle$ or $\langle B', i, j\rangle$, $i, j = 0, 1$. The configuration of the system at step 25 is shown in Figure 6. From Figure 6, it is not difficult to find that, among 16 cells with label 2 there are two cells having objects of those forms. In a cell with label 2 the objects are $\langle A', 0, 1\rangle$, $\langle A', 1, 0\rangle$, in another cell with label 2 the objects are $\langle B', 0, 1\rangle$, $\langle B', 1, 0\rangle$. These objects will be sent to the output cell from cells with label 2. In fact, they represent the same integer number 1.

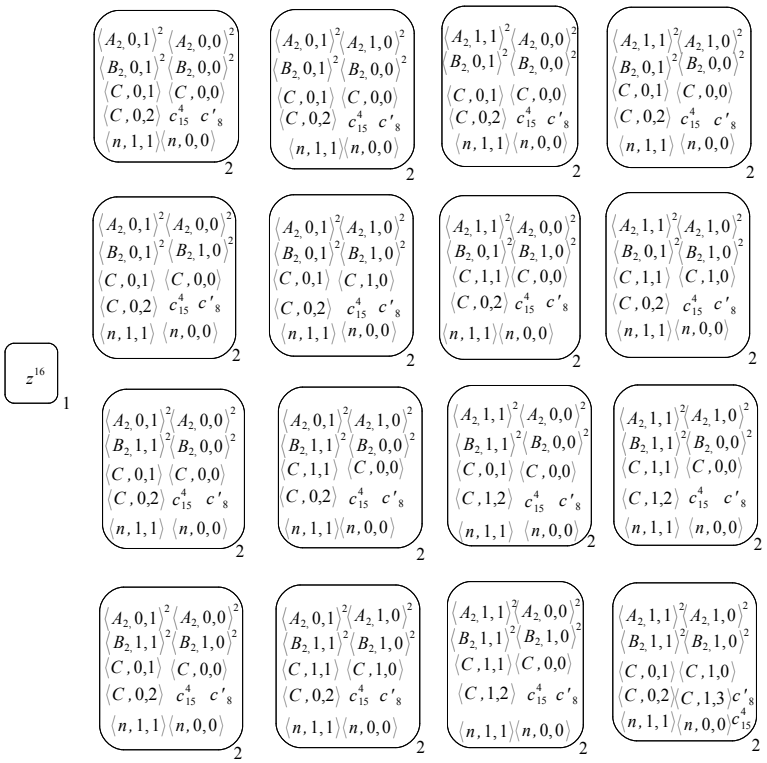


Fig. 4. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 15

4.3 Necessary Resources

From the overview of the computation, it can be found that the family $\{\Pi(k)\}_{k\in\mathbb{N}}$ constructed above can solve the factorization problem in a linear time with respect to the size of the integer to be factored. In what follows, we point out this family of tissue P systems with cell division can be constructed in polynomial time by deterministic Turing machine.

It is easy to check that the rules of a system $\Pi(k)$ of the family are defined recursively from the value k. The necessary resources to build an element of the family are of a polynomial order, as shown below:

- Size of the alphabet: $k^2 + 35k + (4k + 2)\lceil \lg k \rceil + \lceil \lg 2k \rceil + 11 \in O(k^2)$.
- Initial number of cells: $3 \in O(1)$.
- Initial number of objects: $k^2 + 2k + 2 \in O(k^2)$.
- Number of rules: $4k^2 + 39k + \lceil \lg 2k \rceil + (4k + 2)\lceil \lg k \rceil + 4 \in O(k^2)$.
- Maximal length of a rule: $6 \in O(1)$.

Therefore, a deterministic Turing machine can build the tissue P system $\Pi(k)$ in a polynomial time with respect to k.

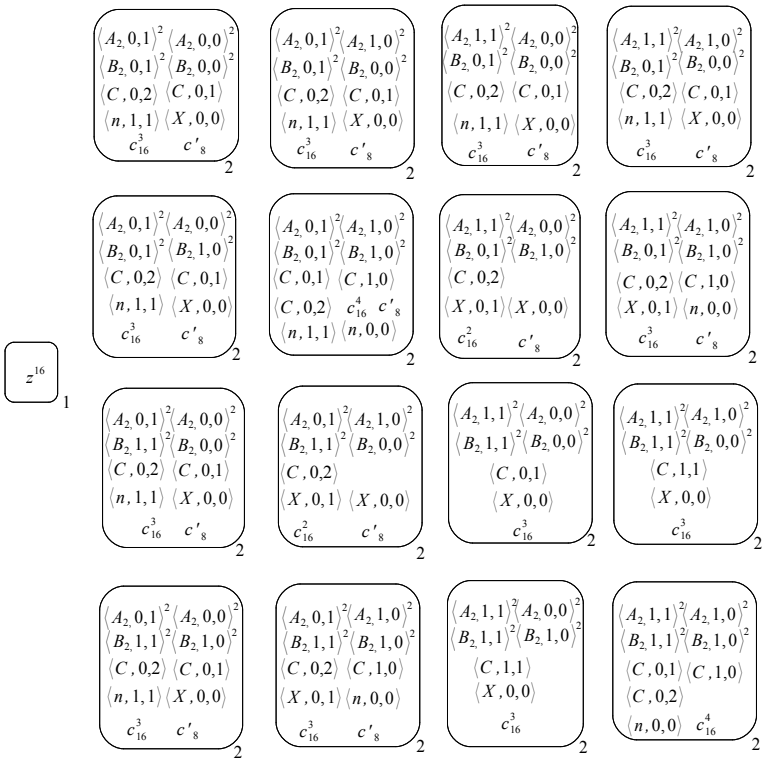


Fig. 5. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 19

5 Conclusions and Comments

Prime factorization problem is not in itself widely useful problem. It has become useful only because it has been found to be crucial for public-key cryptography, and this application is in turn possible only because they have been presumed to be difficult. Currently, no deterministic polynomial-time algorithm is known, which can be executed on Turing machines, that solves the problem for every possible instance. It is of interest to explore any possible and reasonable way to solve prime factorization problem because of its importance in public-key cryptography.

Prime factorization problem is neither decision problem nor optimization problem. In this work, it is considered as a function problem, and in the framework of tissue P systems with cell division, a linear-time solution to prime factorization problem is given. The initial structure of the systems is very simple, which consists of three cells. The system is initialized with inputting into the fixed input cell the multiset that expresses the integer number n to be factored. After a linear time with respect to the size of n (i. e., $\lfloor \lg k \rfloor + 1$), we can read out one factor of n in the output cell.

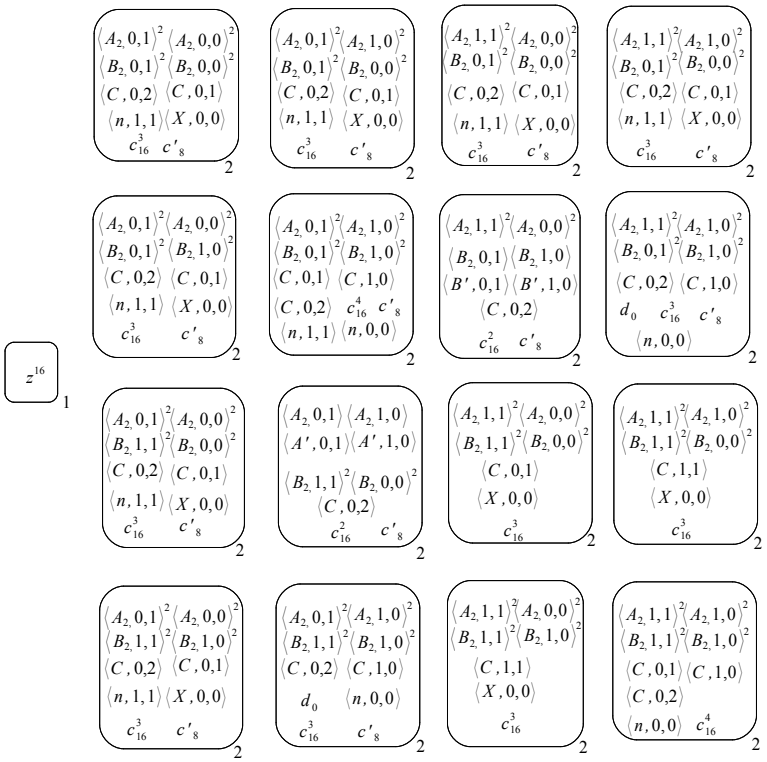


Fig. 6. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 25

P system is a highly distributed parallel model of computation. Currently, nobody knows how to build a biochemical computer/an artificial tissue-like computer. P systems may be implemented using molecules, cells or a large computer network such as the Internet. Although it goes beyond the scope of this work to discuss the implementation of P systems, clearly, it is of particular interest and it is a big challenging topic.

Acknowledgements

The work was supported by National Natural Science Foundation of China (61033003, 61003038 and 30870826), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), Fundamental Research Funds for the Central Universities (2010ZD001), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). Mario J. Pérez-Jiménez also acknowledges the support of the project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds, and the "Proyecto de Excelencia con Investigador de Reconocida Valía" of the Junta de Andalucía under grant P08-TIC04200.

References

1. M. Agrawal, N. Kayal, N. Saxena, PRIMES is in P, *Annals of Mathematics* 160(2) (2004) 781–793.
2. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, A uniform family of tissue P system with cell division solving 3-COL in a linear time, *Theoretical Computer Science* 404 (2008) 76–87.
3. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, Solving subset sum in linear time by using tissue P system with cell division, in: *Lecture Notes in Computer Science*, vol. 4527, 2007, pp. 170–179.
4. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, Computational efficiency of cellular division in tissue-like membrane systems, *Romanian Journal of Information Science and Technology* 11 (3) (2008) 229–241.
5. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, Solving the partition problem by using tissue-like P systems with cell division, in: D. Kearney, V. Nguyen, G. Gioiosa, T. Hendtlass (Eds.), Third International Conference on Bio-Inspired Computing: Theories and Applications, Adelaide, 2008, pp. 43-48.
6. A. Leporati, C. Zandron, G. Mauri, Solving the factorization problem with P systems, *Progress in Natural Science*, 17 (4) (2007) 471–478.
7. A. Leporati, C. Zandron, M.A. Gutiérrez-Naranjo, P systems with input in binary form, *International Journal of Foundation of Computer Science*, 17(1) (2006) 127–146.
8. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón, A new class of symbolic abstract neural nets: tissue P systems, in: *Lecture Notes in Computer Science*, vol. 2387, 2002, pp. 290–299.
9. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón, Tissue P systems, *Theoretical Computer Science* 296 (2003) 295–326.
10. A. Obtulowicz, On P systems with active membranes solving the integer factorization problem in a polynomial time, in: *Lecture Notes in Computer Science*, vol. 2235, 2001, pp. 267–285.
11. A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, *New Generation Computing* 20 (3) (2002) 295–305.
12. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences* 61(1) (2000) 108–143.
13. Gh. Păun, *Membrane Computing. An Introduction*, Springer–Verlag, Berlin, 2002.
14. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, Tissue P system with cell division, *International Journal of Computers, Communications & Control* III (3) (2008) 295–302
15. R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Comunications of the ACM* 21 (2) (2006) 120–126.
16. P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Journal on Computing* 26 (5) (1997) 1484–1509
17. P systems web page `http://ppage.psystems.eu/`

Author Index