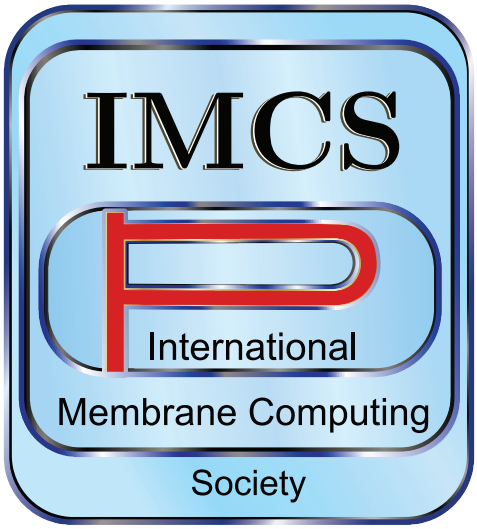
B U L L E T I N

of the International Membrane Computing Society

I M C S



Number 3 June 2017

Bulletin Webpage:

http://membranecomputing.net/IMCSBulletin

Webmaster: Andrei Florea, andrei91ro@gmail.com

Foreword

...And, we have the third issue of our *Bulletin*...

Comparable in the number of pages with the first issue, shorter than the second – maybe this will become a "tradition": the winter volumes to be longer than the summer ones...

Two sections are missing: descriptions of MC research groups and PhD Theses presentations, other sections are rather consistent. It is worth mentioning that we have three areas surveys, which is rather good for the IMCS plans to start editing a journal, to also edit in the near future some collective volumes.

By the way, we also have to applaud the two new books in our area – see the section *Books Announcements* – both of them dealing with applications.

As usual, I would like to stress the fact that the *IMCS Bulletin* is conceived as a working material for the MC community, as a medium for communicating in a fast and efficient way any idea, news, problem, result. As it is already known, each issue gradually grows and remains available at `http://membranecomputing.net/IMCSBulletin`), also being printed. (If somebody wants to have a printed copy, s/he has to contact the IMCS secretary – see information about the structure of IMCS, including email addresses, in the subsequent pages.)

The "instructions to contributors" are minimal. Any material which any MC researcher considers of interest for the community, helping in achieving the goals of IMCS, is very much welcome and can be submitted at any time to the bulletin editor (myself for a while) or to any member of the Bulletin Committee. In what concerns the style and format, the previous issues of the *Bulletin* are available as a model. (If an author needs more precise instructions, please contact me. Standard Latex files are sufficient, LNCS style is the best.)

The copyright of all materials remains with their authors.

I am hereby inviting all people interested in membrane computing to consult the *Bulletin* and, also, to contribute – "views from outside" are always of interest and useful.

*

The realization of the *Bulletin of IMCS* owns very much (i) to all contributors, (ii) to the webmaster, Andrei Florea, andrei91ro@gmail.com, and to the MC research group in Politechnica University in Bucharest, led by prof. Cătălin Buiu, where the bulletin is hosted, and (iii) to prof. Gexiang Zhang, the President of IMCS, to his group, and to the Xihua University in Chengdu, China, where the bulletin is printed.

Gheorghe Păun
June 2017

Contents

Letter from The President 7

IMCS Matters 9

Structure of IMCS 9

Constitution of IMCS 15

IMCS 2016 Prizes 21

IMCS Journal: *Journal of Membrane Computing* (JMEC) 23

Tutorials, Surveys 25

Marian Gheorghe: A Survey of Kernel P Systems 25

Alberto Leporati, Luca Manzoni, Giancarlo Mauri,
Antonio E. Porreca, Claudio Zandron:
Space Complexity of P Systems with Active Membranes:
A Survey 41

Miguel A. Martínez-del-Amor, Agustín Riscos-Núñez,
Mario J. Pérez-Jiménez:
A Survey of Parallel Simulation of P Systems with GPUs 55

Bibliographies 69

Gexiang Zhang, Qiang Yang, Linqiang Pan: A Bibliography
of Technological Applications of Spiking Neural P Systems 69

Book Presentations 75

Andrei George Florea, Cătălin Buiu: Membrane Computing
for Distributed Control of Robotic Swarms 75

Gexiang Zhang, Mario J. Pérez-Jiménez, Marian Gheorghe:
Real-Life Applications with Membrane Computing 77

Open Problems **81**

Zhiqiang Zhang: Some Open Problems on Numerical P Systems with Production Thresholds 81

Bosheng Song: Open Problems on Symport/Antiport (Tissue) P Systems with Channel States 87

Calls for Participation to MC and Related Conferences/Meetings **89**

18th International Conference on Membrane Computing (CMC18), Bradford, UK 89

The 6th Asian Conference on Membrane Computing (ACMC2017) 21-25 September, 2017 Chengdu (P.R. China) 93

15th International Conference on Automata and Formal Languages, Debrecen, Hungary 99

Reports on MC Conferences/Meetings **103**

Report about 15th BWMC 103

Gexiang Zhang, Qiang Yang, Linqiang Pan: A Summary of the Second China Workshop on Membrane Computing (CWMC 2017) 107

Miscellanea **113**

Gheorghe Păun: About the Limits of (Bio)Informatics with Some Illustrations from DNA and Membrane Computing 113

Contents of Previous Volumes 121

Letter from The President

Dear IMCS members,

The first issue of IMCS Bulletin this year is meeting with us. I could not wait to share a piece of good news, namely that IMCS has initiated the first international journal of our research area, *Journal of Membrane Computing (JMEC)*, through a long time of discussion, negotiation and collaboration. More details are also provided in this issue of the Bulletin, in a separate page. All of us are solicited to contribute to and advance this journal.

Eighteen months have passed since our society came into existence. It is perfectly obvious that IMCS has strengthened the collaborations among IMCS members, of which this Bulletin and JMEC are two significant examples. But I would still like to emphasize again the significance and necessity of our collaborations within our society, which is our original intention to create IMCS.

Our society is only a baby who needs to continuously feed for a long time. Your contributions and close collaborations are the required good food and nutrition. Obviously, without a close collaboration, our society could not grow well. Conversely, each of us would be the beneficiary from a good society.

In what follows, let me share with you a delicate point, a topic of debates during recent meetings – it is meant, as the proverb says, "to throw away a brick in order to get a gem". During The Second China Workshop on Membrane Computing this year, a special session was organized to discuss how to write a professional English paper, especially insisting on the issue of how to arrange and cite the related references. In my experience of reviewing conference or journal papers in membrane computing, I observed that some submissions with good novelty and contributions had very few references, even I received some complaints from our P friends that some published journal papers completely neglected others' work to cheat reviewers and readers. The citation was also a discussion during The First China Workshop on Membrane Computing. Except for our collaborated work to advance our society, I hope this is also a point deserving to further discuss and

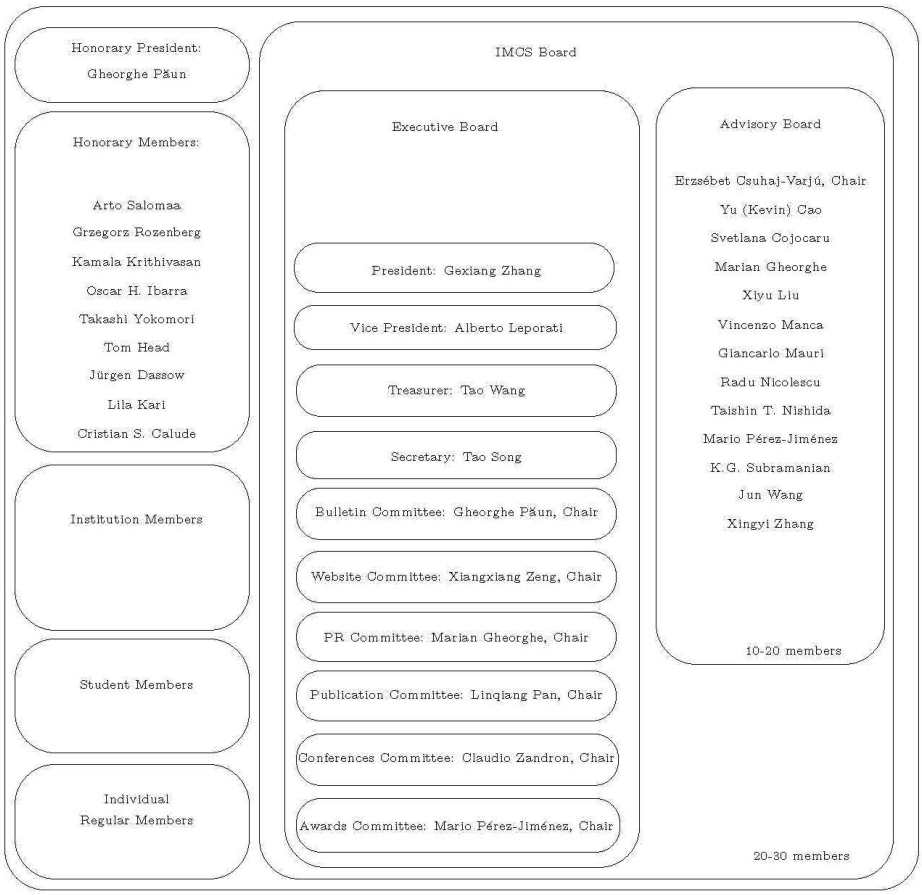
improve through our cooperation. On the eve of running our society journal, such kind of collaborations are particularly important.

Finally, I would like to thank our Bulletin Chair, prof. Gheorghe Păun, and his committee members, and all the contributors for their excellent work and contributions.

Gexiang Zhang
Chengu, China
June 20, 2017

IMCS Matters

Structure of IMCS



The Board of IMCS

The Executive Board:

The Advisory Board:

Honorary President:

Honorary Members:

Bulletin Committee

Website Committee

PR Committee

Marian Gheorghe, Bradford, U.K. – Chair, M.Gheorghe@bradford.ac.uk
Petros Kefalas, Sheffield, U.K. (International Faculty, Greece), kefalas@city.academic.gr
Savas Konur, Bradford, U.K., S.Konur@bradford.ac.uk
Maciej Koutny, Newcastle, U.K., maciej.koutny@newcastle.ac.uk
Jianhua Xiao, Nankai, China, jhxiao@nankai.edu.cn

Publication Committee

Linqiang Pan, Wuhan, China – Chair, lqpanhust@gmail.com
Marian Gheorghe, Bradford, U.K., m.gheorghe@bradford.ac.uk
Alberto Leporati, Milan, Italy, alberto.leporati@unimib.it
Gheorghe Păun, Bucharest, Romania, gpaun@us.es
Mario Pérez-Jiménez, Seville, Spain, marper@us.es
Gexiang Zhang, Chengdu, China, zhgxdylan@126.com

The main tasks of the Publication Committee are (1) to explore the possibility to initiate a series of MC monographs/collective volumes, (2) to establish a MC international journal, (3) to advise the organizers of CMC, ACMC, BWMC, MC workshops in what concerns the special issues of journals, (4) to help translating MC books in Chinese.

The Publication Committee can become the Editorial Board of the MC series of books, but, of course, the journal should have a much larger Editorial Board.

Conferences Committee

Claudio Zandron, Milan, Italy – Chair, zandron@disco.unimib.it
Henry Adorna, Quezon City, Philippines
Artiom Alhazov, Chişinău, Rep. of Moldova
Bogdan Aman, Iaşi, Romania
Matteo Cavaliere, Edinburgh, Scotland
Erzsébet Csuhaj-Varjú, Budapest, Hungary
Rudolf Freund, Wien, Austria
Marian Gheorghe, Bradford, U.K. – Honorary Member
Thomas Hinze, Cottbus, Germany
Florentin Ipate, Bucharest, Romania
Shankara N. Krishna, Bombay, India
Alberto Leporati, Milan, Italy
Taishin Y. Nishida, Toyama, Japan

Linqiang Pan, Wuhan, China – responsible of ACMC
Gheorghe Păun, Bucharest, Romania – Honorary Member
Mario J. Pérez-Jiménez, Sevilla, Spain
Agustín Riscos-Núñez, Sevilla, Spain
Petr Sosík, Opava, Czech Republic
K.G. Subramanian, Chennai, India
György Vaszil, Debrecen, Hungary
Sergey Verlan, Paris, France
Gexiang Zhang, Chengdu, China

Awards Committee:

Mario Pérez-Jiménez, Seville, Spain – Chair, marper@us.es
Marian Gheorghe, Bradford, U.K., m.gheorghe@bradford.ac.uk
Giancarlo Mauri, Milan, Italy, mauri@disco.unimib.it
Gheorghe Păun, Bucharest, Romania, gpaun@us.es
Linqiang Pan, Wuhan, China, lqpanhust@gmail.com

Rules of functioning:

1. Prizes to be awarded annually: (1) The PhD Thesis of the Year, (2) The Theoretical Result of the Year, (3) The Application of the Year.
2. Each prize consists of diplomas for each co-author, one copy of the Hamangia thinker[1] and one voucher for a discount in the registration fee for the first of BWMC, CMC or ACMC to take place after the prize was voted; the discount will be fixed by the organizing committee of the meeting; in case of several authors, they will choose the one of them to benefit of the voucher.
3. Any registered member of IMCS can be nominated and can receive any of the three prizes. In cases (2) and (3), the prizes are awarded to the authors of a paper or of an application, with at least one of authors being a member of IMCS. The members of the Awards Committee cannot receive any prize, neither they can be coauthors of papers or applications which receive one of prizes (2) and (3).
4. If the Awards Committee considers necessary, each year at most one of the prizes can be awarded ex aequo, to two winners.
5. Any registered member of IMCS can propose a candidate for any prize, by sending to any member of the Awards Committee the relevant information (and any additional information requested by the Awards Committee). Implicitly, the Awards Committee can itself make nominations.

[1] A Neolithic age clay sculpture, about 4000 years BC, found in Romania – see the image at the next page.

6. The nominations for the year Y should reach the Awards Chair before 20 of January of the year Y+1. The Awards Committee members decide the winners by the middle of February, and the prizes are awarded at the first edition of BWMC, CMC or ACMC where the winners participate in.
7. The members of the Awards Committee and the rules of functioning can be changed every year, after March 1, at the proposal of the Chair person or of any member of the IMCS Board, subject to a vote in the IMCS Board.

The IMCS prizes are mainly meant to reward the excellency in MC research, equally focusing on theory and applications, and to encourage young researchers.

The prizes are not subject to competitions, they do not identify *the best* PhD thesis or paper or application, they just point that a certain work/result is of a high value. This does not imply that other works/results are not so. We cannot rank scientific results like in sport, in a mathematical sense.

We only want to call attention to certain works – thus also calling attention to MC and to IMCS.

The prestige of a prize will be given by the prestige of the winners, also on their evolution in time, during their careers.

To reach these goals, we have to be conservative, exigent, transparent in our nominations and, especially, in selection.

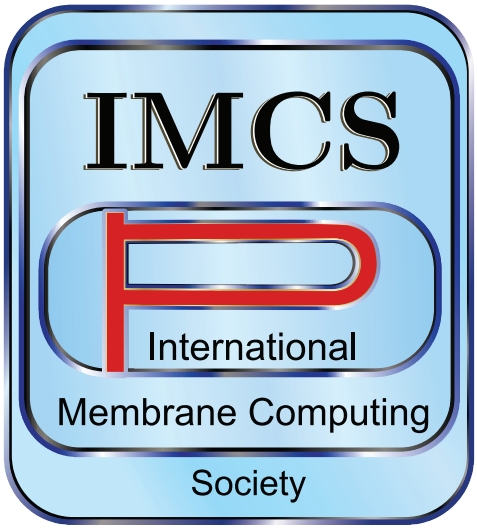
Nominations for the prizes for 2017 are waited for until January 20, 2018, and can be sent at any time, electronically, to any member of the Awards Committee (preferably with a CC to all members).

Constitution of the International Membrane Computing Society – IMCS

Article 1: Name

(1.1.) The name of the Society shall be International Membrane Computing Society, abbreviated IMCS.

(1.2.) The logo of IMCS is the one in the figure below. It should appear in all relevant places, such as IMCS web page, posters, calls, on the cover of the *Bulletin of IMCS*, etc.



Article 2: Objects

(2.1.) The society shall be a nonprofit academic organization, having as its goal to promote the development of membrane computing (MC), internationally, at all levels (theory, applications, software, implementations, connection with related areas, etc.).

(2.2.) A special attention will be paid to the communication/cooperation inside MC community, to connections with other professional scientific organizations with similar aims, and to promoting MC to young researchers.
(2.3.) IMCS will publish proceedings, journals or other materials, printed or electronically, as it sees fit.
(2.4.) IMCS will organize yearly MC meetings, such as the **Conference on Membrane Computing** (CMC), the **Asian Conference on Membrane Computing** (ACMC), the **Brainstorming Week on Membrane Computing** (BWMC), as well as further workshops/meetings, as it sees fit.

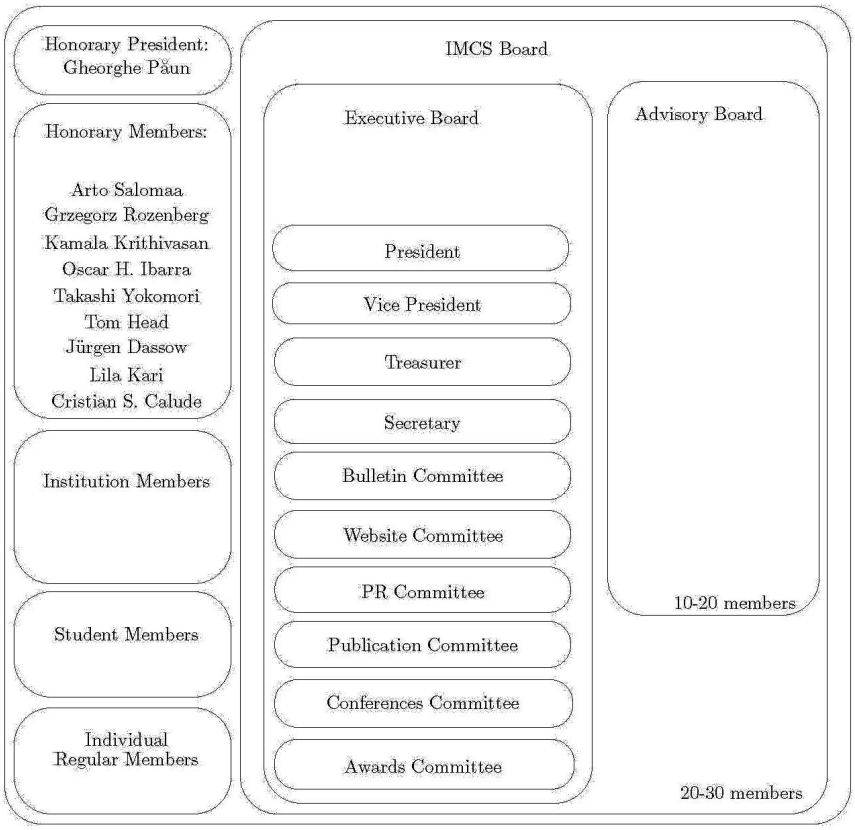
Article 3: Membership

(3.1.) There are four categories of members: **Honorary Members, Regular Members, Student Members**, and **Institution Members**.
(3.2.) The Honorary Members are elected by the IMCS Board (email voting, majority rule). Regular membership is open to all persons interested, on completing a membership form.
(3.3.) Student Members can be undergraduate, master, and PhD students, and they are eligible for various facilities IMCS is planning for students.
(3.4.) Institutions which want to join IMCS and support it can become Institution Members. Any support/sponsorship from an institution will be acknowledged in an appropriate way in IMCS publications.
(3.5.) Any member, of any kind, is supposed to know and accept the Constitution of IMCS.

Article 4: Structure

(4.1.) The structure of IMCS and its governance are as specified in the next figure. The figure also specifies the ten Honorary Members with whom the Society starts (February 2016).
(4.2.) Gheorghe Păun, the founder of MC, is appointed **Honorary President** of IMCS.
(4.3.) The work of IMCS is organized and conducted by the **Board of IMCS**, consisting of the **Executive Board** and the **Advisory Board**. The Advisory Board should have between 10 and 20 members, hence in total the IMCS Board should contain between 20 and 30 members,
(4.4.) The Executive Board consists of four individual positions: **President, Vice President, Treasurer**, and **Secretary**, and six **Committees: Bulletin, Website, PR, Publication, Conferences**, and **Awards Committee**. Each of these six Committees has a chair person. The Advisory Board also has a chair person.
(4.5.) The four individual positions from the Executive Board, the six chair persons of the Committees, the members of the Advisory Board, and the chair of the Advisory Board are elected by the IMCS Board (email voting, majority rule). Each chair of a Committee appoints a number of Committee members as he/she sees fit.

(4.6.) All the elected positions are elected for one year. After one year, a change of an elected person can be proposed by the President or the Vice President of the IMCS Board, by the person itself (resignation), or by two thirds of members of the IMCS Board, and it is voted in the IMCS Board (email voting, majority rule). If there is no change proposal, then the person who occupies any position in the IMCS Board continues in the same position, for one further year.



Article 5: Duties and competencies

(5.1.) The IMCS Board President represents the Society in relation with any external entity, organizes/coordinates the activity of the IMCS Board, initiates voting in the IMCS Board, chairs any panel/meeting of the Society.

(5.2.) The Vice President helps the President in all his/her activity, represents the President when he/she is not available (e.g., in chairing panels/meetings). Every year, the President and the Vice President present a common report about IMCS activity, first circulating it by email in the IMCS Board and, after possible corrections, posting it in the IMCS web page.

(5.3.) The Treasurer takes care of the income and expenses of IMCS, and each year presents a report in this respect to IMCS Board. This report is analyzed and voted in the IMCS Board (email voting, majority rule).
(5.4.) The Secretary is responsible to keep a track record of the IMCS: memberships, reports, voting results, etc.
(5.5.) The Bulletin Committee takes care of editing the *Bulletin of IMCS*, first accumulating information/materials in an electronic format and then printed, if needed/requested, with a periodicity to be decided by the IMCS Board.
(5.6.) The Website Committee takes care of the IMCS web page, whose structure should be decided by the IMCS Board.
(5.7.) The PR Committee is responsible with developing relationships with other similar organizations and promoting IMCS on various scientific forums, advertising its activity on specialised networks and at international events.
(5.8.) The Publication Committee supervises the publication of proceedings, special issues of journals, collective volumes edited under the auspices of IMCS. Two particular goals of this Committee are to initiate a specialized journal, *International Membrane Computing Journal*, and a specialized series of monographs.
(5.9.) The Conference Committee works as a steering committee for the two MC yearly conferences, CMC and ACMC, looking for venues, suggesting (in cooperation with the organizing committees) program committees and invited speakers, possible sponsors and publications.
(5.10.) The Awards Committee collects nominations and decides the winners of three yearly IMCS Prizes: **(1) The PhD Thesis of the Year, (2) The Theoretical Result of the Year, (3) The Application of the Year**. The Awards Committee has its Rules of functioning, which are voted by the IMCS Board (email voting, majority rule).

Article 6: Voting

(6.1.) Each member of the IMCS Board (between 20 and 30 members) has one vote.
(6.2.) A voting, on any subject, can be initiated by the President, the Vice President, or by two thirds of the IMCS Board members.
(6.3.) The message proposing a vote should specify the issue to be decided in such a way that the alternatives YES and NO are clear. The message should be sent to all members of the IMCS Board, the voting messages of the members should also be sent to all members (full transparency). The voting should last 30 days. If a member is not replying in the first 15 days, the initiator of the vote should contact him/her once again. If a member is not replying even to the second message, then his/her vote is considered *abstaining*.
(6.4.) "Majority rule" means that at least half of the IMCS Board have voted (YES, NO, or abstaining) and the decision is made according to the number of YES and NO votes which is higher. In case of a draw, the vote of the President is decisive – unless if the President does not decide to repeat the vote, maybe changing the object of the vote.

(6.5.) All ambiguities and uncovered cases should be clarified by discussions in the IMCS Board and, if decided so, proposed as amendments to the Constitution.

Article 7: Panels

(7.1.) On the occasions of IMCS annual meetings, like BWMC, CMC, and ACMC, panels should be organized, chaired by the President, the Vice President or, in their absence, by another member of the IMCS Board designated by the President, to discuss current issues of the Society.

Article 8: Finance

(8.1.) Income: possible membership fees, as decided by IMCS Board, donations, sponsorhips, conference registration fees, participation to research projects.

(8.2.) Expenses: IMCS prizes, students support, *Bulletin of IMCS* hardcopy, maintaining web pages, sponsoring MC conferences – all these and anything else, under the control of the IMCS Board.

Article 9: Amendments

(9.1.) Amendments to IMCS Constitution can be proposed by any member of the IMCS Board, at any time. Any amendment should be discussed and voted in the IMCS Board (email voting, majority rule) and then, if accepted, published in the IMCS web page, thus being available to all members of IMCS.

Article 10: Dissolution

(10.1.) The dissolution of IMCS should be done in two steps: first, a vote in the IMCS Board is organized (email voting, two thirds majority), and, if the dissolution proposal passes, a general vote is organized, where all regular members participate (email voting, two thirds majority; in order the vote to be valid, at least half of the members should vote).

(10.2.) If the Society decides to get dissolved, all remaining assets shall be donated to a similar organization, at the choice of the IMCS Board.

Article 11. Provisory statement

The present Constitution will get provisionally valid by being voted (by email, majority rule), in March 2016, in the IMCS Board, as this Board was constituted by consensus during BWMC 2016 and soon after that. Then, it will be published in the IMCS web page and, as soon as possible, in 2016, it will be voted by all individual members of IMCS (email voting, majority rule). The vote will last one month and to voting will participate all individual members of IMCS, students or regular, registered until the last day of the previous month.

IMCS 2016 Prizes

1 Nominations

The IMCS Awards Committee members have selected the following final list of candidates:

The PhD Thesis of the Year 2016

(a) Author: Suresh Kumar
Title: *Two Dimensional P Systems: Arrays, Graphs and Applications*
Supervisor: R. Rama
Thesis defense: September 16, 2016

(b) Author: Tao Wang
Title: *Spiking Neural P Systems and Their Applications in Fault Diagnosis of Electric Power Systems*
Supervisor: Gexiang Zhang
Thesis defense: November 8, 2016
Graduation certificate: December 2016

The Theoretical Result of the Year

(a) Title: On the Computational Power of Spiking Neural P Systems with Self-Organization
Authors: Xun Wang, Tao Song, Faming Gong, Pan Zheng
Journal: *Scientific Reports*

(b) Title: Generalized P Colonies with Passive Environment
Authors: L. Ciencialová, L. Cienciala, P. Sosík
Journal: *Theoretical Computer Science* and *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*

(c) Title: On the Classes of Languages Characterized by Generalized P Colony Automata
Authors: K. Kántor, G. Vaszil
Journal: *Theoretical Computer Science* and *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*

The Application of the Year

(a) Title: Complex Network Clustering by a Multi-objective Evolutionary Algorithm Based on Decomposition and Membrane Structure
Authors: Y. Ju, S. Zhang, N. Ding, X. Zeng, X. Zhang
Journal: *Scientific Reports*
(b) Title: Building a Basic Membrane Computer
Authors: A. Millán, J. Viejo, J. Quiros, M.J. Bellido, D. Guerrero, E. Ostua
Journal: *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*
(c) Title: Improving simulations of Spiking Neural P Systems in NVIDIA CUDA GPUs: CuSNP
Authors: J.P. Carandang, J. Matthew, B. Villa Ores, F.G.C. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor
Journal: *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*

2 Winners

The IMCS Awards Committee has decided the following:

- No prize for the best PhD thesis is awarded for 2016.
- For the theoretical result of the year, the prize goes ex aequo, to (b) and (c), that is to the papers by
  1. L. Ciencialová, L. Cienciala, P. Sosík: Generalized P Colonies with Passive Environment
  2. K. Kántor, G. Vaszil: On the Classes of Languages Characterized by Generalized P Colony Automata.
- For the application of the year, the prize also goes ex aequo to (a) and (c), that is to the papers by
  1. Y. Ju, S. Zhang, N. Ding, X. Zeng, X. Zhang: Complex Network Clustering by a Multi-objective Evolutionary Algorithm Based on Decomposition and Membrane Structure
  2. J.P. Carandang, J. Matthew, B. Villa Ores, F.G.C. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor: Improving Simulations of Spiking Neural P Systems in NVIDIA CUDA GPUs: CuSNP.

Nominations for the prizes for 2017 are waited for until January 20, 2018.

IMCS Journal:
Journal of Membrane Computing
(JMEC)

International Membrane Computing Soceity (IMCS) witnesses the processes of gestation and the birth of the new international journal, *Journal of Membrane Computing (JMEC)*. The Editor-in-Chief of JMEC, Professor Linqiang Pan, signed the publishing agreement with Springer Nature Singapore Pte Ltd. on March 20, 2017, in Wuhan, China.

JMEC is an international journal with four issues per volume (per year). The Journal Homepage *http://www.springer.com/journal/41965* is setting up and will be publicized soon. The first accepted papers are planned to be online in 2018 and thereafter will be published in the Spring of 2019.

JMEC aims to foster the dissemination of new discoveries and novel technologies in the area of membrane computing and the related areas like bio-inspired computing and natural computing. The focus of this journal is to provide a publication and communication platform for researchers, professionals and industrial practitioners, covering the theoretical fundamentals and technological advances to various applications. JMEC solicits original, high-quality and previously unpublished research papers, survey and review articles, short communications, and tutorial papers.

Tutorials, Surveys

A Survey of Kernel P Systems

Marian Gheorghe

School of Electrical Engineering and Computer Science, University of Bradford
Bradford BD7 1DP, UK
m.gheorghe@bradford.ac.uk

1 Introduction

The membrane systems concept has been introduced through the seminal paper [23] and has then intensively studied and enriched with various new features and research topics. Soon after its launch, the area has started growing very fast, a research monograph being published [24] and then later on a handbook presenting the key developments in this area [25]. The research problems envisaged to be investigated have been outlined, a few years ago, in a paper looking at the frontiers of this field [13]. The membrane computing area includes now a wealth of computational models studied for their computational power and efficiency, but also investigated for their usage in various applications [25].

One such membrane systems model is the kernel P systems device. This has been introduced in [6] and its definition revised in [7]. The kernel P systems research covers a broad spectrum of topics, from theory to verification, applications and simulations. Here we present a summary of the research results in this area.

In the sequel we introduce the key concepts, present some relationships between P systems with active membranes and electrical charges, and kernel P systems, introduce verification and testing capabilities of these models, and finish off the presentation with a brief overview of the applications of the kernel P systems.

2 kP Systems - Main Concepts and Definitions

The kernel P systems (kP systems, for short) will be defined below. For standard P systems concepts and results we refer to [24, 25].

The kP systems concepts and definitions used in this paper are from [6, 7]. We start with some preliminary concepts.

2.1 Preliminaries

For a finite alphabet $A = \{a_1, ..., a_m\}$, A^* denotes the set of all strings (sequences) over A. The empty string is denoted by λ and $A^+ = A \setminus \{\lambda\}$ denotes the set of non-empty strings. For a string $u \in A^*$, $|u|_a$ denotes the number of occurrences of a in u, where $a \in A$. For a subset $S \subseteq A$, $|u|_S$ denotes the number of occurrences of the symbols from S in u. The length of a string u is given by $\sum_{a_i \in A} |u|_{a_i}$. The length of the empty string is 0, i.e. $|\lambda| = 0$. A multiset over A is a mapping $f : A \to \mathbb{N}$. Considering only the elements from the support of f (where $f(a_{i_j}) > 0$, for some j, $1 \leq j \leq p$), the multiset is represented as a string $a_{i_1}^{f(a_{i_1})} \dots a_{i_p}^{f(a_{i_p})}$, where the order is not important. In the sequel multisets will be represented by such strings.

2.2 kP Systems Basic Definitions

We start by introducing the concept of a *compartment type* utilised later in defining the compartments of a kP system.

Definition 1. *T is a finite* set of compartment types, *$T = \{t_1, \dots, t_s\}$, where $t_i = (R_i, \sigma_i)$, $1 \leq i \leq s$, consists of a set of rules, R_i, over an alphabet A, and an execution strategy, σ_i, defined over $Lab(R_i)$, the labels of the rules of R_i.*

Now we formally define a kP system and the compartments that appear in this definition are constructed using compartment types introduced by Definition 1. Each such compartment, C, will be defined by a tuple (t, w), where $t \in T$ is the type of the compartment and $w \in A^*$ the initial multiset of it. The other concepts that appear in this definition, the types of rules and the execution strategies, will be introduced later.

Definition 2. *A* kP system *of degree n is a tuple*

$$k\Pi = (A, \mu, C_1, \dots, C_n, i_0),$$

where A is a finite set (an alphabet) of elements called objects*; μ defines the* initial membrane structure*, which is a graph, (V, E), where V is a finite set of vertices indicating compartments of $k\Pi$, and E a finite set of edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a* compartment *of the kP system; i_0 is the label of the* output compartment*, where the result is obtained.*

2.3 kP Systems Rules

Each rule of a kP system has the form $r\ \{g\}$, where r stands for the rule itself and g is its guard. The guards are constructed using multisets over A, as operands, and relational and/or Boolean operators. We denote by *Rel* the set of relational operators, $\{<, \leq, =, \neq, \geq, >\}$, and let $\gamma \in Rel$, a relational operator, and for $a \in A$, a^n a multiset. We first introduce an *abstract relational expression*.

Definition 3. *Let g be the* abstract relational expression *denoted γa^n and w a multiset; then the guard g applied to w denotes the* relational expression *$|w|_a \gamma n$.*

The abstract relational expression g is true for the multiset w, if $|w|_a \gamma n$ is true.

For the Boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction), we consider an *abstract Boolean expression* as being defined by one of the following conditions

- any abstract relational expression is an abstract Boolean expression;
- if g and h are abstract Boolean expressions then $\neg g$, $g \wedge h$ and $g \vee h$ are abstract Boolean expressions.

Definition 4. *Let g be an* abstract Boolean expression *containing g_i, $1 \leq i \leq q$, abstract relational expressions and w a multiset; then g applied to w means the* Boolean expression *obtained from g by applying g_i, $1 \leq i \leq q$, to w.*

As in the case of an abstract relational expression, the guard g is true with respect to the multiset w, if the abstract Boolean expression g applied to w is true.

Example 1. If g is the guard defined by the abstract Boolean expression $\geq a^5 \wedge < b^3$ and w a multiset, then g applied to w is true if w has at least 5 a's and no more than 2 b's.

We now introduce the types of rules allowed in a kP system.

Definition 5. *A rule from a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$ can have one of the following types:*

- *(a)* **rewriting and communication** *rule: $x \to y\ \{g\}$, where $x \in A^+$ and y has the form $y = (a_1, t_1) \ldots (a_h, t_h)$, $h \geq 0$, $a_j \in A$, and t_j, $1 \leq j \leq h$, indicates the type of a compartment linked to the current one; t_j might also indicate the type of the current compartment, C_{l_i}; if a link between C_{l_i} and a compartment of type t_j does not exists (i.e., there is no corresponding edge in E) then the rule is not applied; if the type t_j appears in more than one compartment connected to C_{l_i}, then one of them will be non-deterministically chosen;*
- *(b)* **structure changing rules***; the following types of rules are considered:*
  - *(b1)* **membrane division** *rule: $[x]_{t_{l_i}} \to [y_1]_{t_{i_1}} \ldots [y_p]_{t_{i_p}}\ \{g\}$, where $x \in A^+$ and $y_j \in A^*$, $1 \leq j \leq p$; the compartment C_{l_i} will be replaced by p compartments; the j-th compartment, $1 \leq j \leq p$, of type t_{i_j} contains the same objects as C_{l_i}, but x, which will be replaced by y_j; all the links of C_{l_i} are inherited by each of the newly created compartments;*
  - *(b2)* **membrane dissolution** *rule: $[\ ]_{t_{l_i}} \to \lambda\ \{g\}$; the compartment C_{l_i} will be destroyed together with its links;*
  - *(b3)* **link creation** *rule: $[x]_{t_{l_i}}; [\ ]_{t_{l_j}} \to [y]_{t_{l_i}} - [\ ]_{t_{l_j}}\ \{g\}$, where $x \in A^+$ and $y \in A^*$; the current compartment is linked to a compartment of type t_{l_j} and x is transformed into y; if more than one compartment of type t_{l_j} exist and they are not linked with $C_{t_{l_i}}$, then one of them will be non-deterministically picked up; g is a guard that refers to the compartment of type t_{l_i};*

– *(b4)* **link destruction** *rule:* $[x]_{t_{l_i}} - [\ ]_{t_{l_j}} \rightarrow [y]_{t_{l_i}}; [\ ]_{t_{l_j}}\ \{g\}$*, where* $x \in A^+$ *and* $y \in A^*$*; is the opposite of link creation and means that the compartments are disconnected.*

When in a rewriting and communication rule one of the right hand side elements (a_j, t_j), $1 \leq j \leq h$, is such that $t_j = t_{l_i}$, then this is simply written as a_j.

2.4 kP Systems Execution Strategies

Each compartment of a kP system has its own execution strategy, in accordance with each compartment type t from T – see Definition 1. As in Definition 1, $Lab(R)$ is the set of labels of the rules R.

Definition 6. *For a compartment type* $t = (R, \sigma)$ *from* T *and* $r \in Lab(R)$*,* $r_1, \ldots, r_s \in Lab(R)$*, the* execution strategy*,* σ*, is defined by the following*

- $\sigma = \lambda$*, means no rule from the current compartment will be executed;*
- $\sigma = \{r\}$ *– the rule* r *is executed;*
- $\sigma = \{r_1, \ldots, r_s\}$ *– one of the rules labelled* $r_1, \ldots, r_s$ *will be non-deterministically chosen and executed; if none is applicable then nothing is executed; this is called* alternative *or* choice*;*
- $\sigma = \{r_1, \ldots, r_s\}^*$ *– the rules are applied an arbitrary number of times* (arbitrary parallelism*);*
- $\sigma = \{r_1, \ldots, r_s\}^\top$ *– the rules are executed according to the* maximal parallelism *strategy;*
- $\sigma = \sigma_1 \& \ldots \& \sigma_s$*, means executing sequentially* $\sigma_1, \ldots, \sigma_s$*, where* σ_i*,* $1 \leq i \leq s$*, describes any of the above cases; if one of* σ_i *fails to be executed then the rest is no longer executed;*
- *for any of the above* σ *strategy only one single structure changing rule is allowed.*

A *configuration* of a kP system with n compartments, $C_1, \ldots, C_n$, is a tuple $c = (u_1, \ldots, u_n)$, where u_i is a multiset belonging to compartment C_i, $1 \leq i \leq n$. Structure changing rules might be applied and the number of compartments will change. A configuration $c' = (v_1, \ldots, v_m)$ is obtained in one step from $c = (u_1, \ldots, u_n)$, written as $c \Longrightarrow c'$, if in each compartment C_i the execution strategy σ_i is applied to u_i, $1 \leq i \leq n$. A *computation*, as usual in membrane computing, is defined as a finite sequence of steps starting from the initial configuration, $(w_1, \ldots, w_n)$, and applying in each step and each compartment the rules of the corresponding execution strategy.

Remark 1. The result of a computation will be the number of objects collected in the output compartment. For a kP system, $k\Pi$, the set of all these numbers will be denoted by $M(k\Pi)$.

Example 2. We generate square numbers starting with 1 by using the kP system, $k\Pi_{sq}$, having $T = \{t_1, t_2\}$, where $t_j = (R_j, \sigma_j)$ (R_j and σ_j are defined below), $1 \leq j \leq 2$

$$k\Pi_{sq} = (A, \mu, C_1, C_2, 2),$$

with

- the alphabet $A = \{a, s, b, t\}$;
- μ is the graph with nodes C_1, C_2 and the edge linking them;
- $C_j = (t_j, w_j)$, $1 \leq j \leq 2$, where $w_1 = a$ and $w_2 = \lambda$;
- C_2 is the output compartment.

The set of rules R_1 contains
$r_1 : a \to \lambda \ \{= t\}$,
$r_2 : s \to (s, 2) \ \{= t\}$,
$r_3 : a \to ab^2s \ \{< t\}$,
$r_4 : a \to ast \ \{< t\}$,
$r_5 : b \to bs \ \{< t\}$;
and $R_2 = \emptyset$. The execution strategy is $\sigma_1 = R_1^{\top}$ for t_1; σ_2 may be any execution strategy.

In n, $n \geq 1$, steps one can obtain $(a, \lambda) \Longrightarrow \ldots \Longrightarrow (ab^{2(n-1)}ts^{n^2}, \lambda)$ by using $(n-1)$ times the rules r_3 and r_5 and once the rule r_4. In the next step one can get $(b^{2(n-1)}t, s^{n^2})$, by using r_1, r_2. In each step the rules mentioned above are applied in the maximal parallel manner. In $(n+1)$ steps in C_2 is obtained the square of n, i.e., s^{n^2}.

3 kP Systems and P Systems with Active Membranes

The kP systems introduced above represent a class of computational models including various features of existing P systems as well as new ones. It is expected that the computational power equals that of a Turing machine. We are more interested to show how the behaviour of other P systems can be simulated by specific kP systems. Showing this, we not only prove the computational completeness of this new model, but also demonstrate its effectiveness in simulating the behaviour of other classes of P systems. In [18] has been shown how a generalised communicating P systems problem is mapped into its kP systems based specification. In [7, 8, 4] it has been shown how P systems with active membranes using electrical charges and neural-like P systems can be simulated by kP systems. In the sequel we present the connection between P systems with active membranes having electrical charges and kP systems, with the proof given in [4]. Beforehand we introduce P systems with active membranes and electrical charges.

Definition 7. *A P system with active membranes of initial degree n is a tuple (see [25], Chapter 11)* $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n,0}, R, i_0)$ *where:*

- *O is an alphabet of objects, $w_{1,0}, \ldots, w_{n,0}$ are the initial strings in the n initial compartments and i_0 is the output compartment;*
- *H is the set of labels for compartments;*
- *μ defines the tree structure associated with the system;*
- *R consists of rules of the following types:*
  - *(a)* rewriting rules*: $[u \to v]_h^e$, for $h \in H$, $e \in \{+, -, 0\}$ (set of electrical charges), $u \in O^+$, $v \in O^*$;*
  - *(b)* send-in communication rules*: $u[\ ]_h^{e_1} \to [v]_h^{e_2}$, for $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $u \in O^+$, $v \in O^*$;*
  - *(c)* send-out communication rules*: $[u]_h^{e_1} \to [\ ]_h^{e_2} v$, for $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $u \in O^+$, $v \in O^*$;*
  - *(d)* dissolution rules*: $[u]_h^e \to v$, for $h \in H \backslash \{s\}$, s denotes the skin membrane (the outmost one), $e \in \{+, -, 0\}$, $u \in O^+$, $v \in O^*$;*
  - *(e)* division rules for elementary membranes*: $[u]_h^{e_1} \to [v]_h^{e_2}[w]_h^{e_3}$, for $h \in H$, $e_1, e_2, e_3 \in \{+, -, 0\}$, $u \in O^+$, $v, w \in O^*$.*

The rules are executed in accordance with the maximal parallelism, but in each compartment only one of the rules (b)-(e) is executed. In the sequel we assume that the output compartment is neither dissolved nor divided. The result of a computation, obtained in i_0, is denoted by $M(\Pi)$.

We consider a particular case of a P system with active membranes, namely a P system with active membranes starting with n_1 compartments and having an upper bound limit for the number of active compartments. We will show that for every multiset w obtained in the output compartment of a P system with active membranes satisfying the above conditions, Π, there is a kP system, $k\Pi$, such that there is $e \in \{+, -, 0\}$ and we is obtained in the output compartment of $k\Pi$.

Theorem 1. *If Π is a P system with active membranes having n_1 initial compartments and an upper bound for the number of active compartments in any computation, then there exists a kP system, $k\Pi$, of degree 2 and using only rewriting and communication rules, such that $x + 1 \in M(k\Pi)$, iff $x \in M(\Pi)$.*

Proof. Let $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n_1,0}, R, i_0)$ be a P system with active membranes of initial degree n_1, and the polarisations of the n_1 compartments are all 0, i.e., $e_1 = \cdots = e_{n_1} = 0$.

We will build a kP system with two compartments, C_1, where the behaviour of Π will be simulated, and C_2, associated with i_0.

The dynamic structure of the P system Π will be handled be using the following mechanisms. Each membrane will be identified by a pair (i, h), where $i \in I$ is an index associated with the membrane and $h \in H$ is its label. We work under the assumption that I is finite. Its cardinal is equal to the maximum number of active membranes that may appear in any computation. We let $i_0 \in I$ and $i_0 \in H$. We will denote by $(I \times H)_c$ the currently used pairs (i, h). We assume that for any $(i, h) \in (I \times H)_c$ and $(j, h') \in (I \times H)_c$, we have $i \neq j$, hence the cardinal of $(I \times H)_c$ is always at most the cardinal of I. Whenever a membrane dissolution takes place,

its index and label are removed from $(I \times H)_c$. When a membrane division rule is applied, the index and label of the divided compartment are removed from $(I \times H)_c$ and two new values of indices with the same label are selected and added to the set $(I \times H)_c$. The tuple (i_0, i_0) is always in $(I \times H)_c$.

A compartment of Π of label h, electrical charge e and containing the multiset w will be denoted by $[w]_h^e$. We will codify a compartment $[w]_h^e$ by two tuples $< e, i, h >$ and $< w, i, h >$, with $(i, h) \in (I \times H)_c$. For a multiset $w = a_1 \dots a_m$, $< w, i, h >$ denotes $< a_1, i, h > \cdots < a_m, i, h >$. For a compartment $[w]_h^e$, when $h \neq i_0$, the tuples $< e, i, h >$ and $< w, i, h >$ are added to C_1; when $h = i_0$ then in addition to the tuples present in C_1, we add e and w to C_2.

For $(i, h) \in I \times H$ we denote by $p(i, h)$ the parent of the membrane with label h and of index i. If $p(i, h) = (i', h')$ then the membrane with label h' and index i' is the parent of the membrane with label h and index i. By $< x, p(i, h) >$ and $< e, p(i, h) >$ we denote the tuples $< x, i', h' >$ and $< e, i', h' >$, respectively.

Two new symbols, δ_1 and δ_2, will be used for the membrane dissolution and division. The following guard will be used

$$\overline{= \delta_{all}} := \bigwedge_{(i,h) \in I \times H} (\neg =< \delta_1, i, h >) \wedge (\neg =< \delta_2, i, h >),$$

which is true for a multiset w when none of $< \delta_j, i, h >$, $1 \leq j \leq 2$, $i \in I$, and $h \in H$, appears in w. We also introduce a guard checking that the symbols γ_1 and γ_2 do not appear in the current multiset:

$$\overline{= \gamma_{all}} := (\neg = \gamma_1) \wedge (\neg = \gamma_2).$$

We construct the kP system, $k\Pi$, using $T = \{t_1, t_2\}$, where $t_j = (R'_j, \sigma_j)$ (with R'_j and σ_j being defined later), $1 \leq j \leq 2$, as follows:

$$k\Pi = (A, \mu', C_1, C_2, 2),$$

with the elements of the system given below

- μ' is the graph with nodes C_1, C_2 and the edge linking them;
- the alphabet is
  $A = O \cup \{+, +', -, -', 0, 0', \gamma_1, \gamma_2\} \cup \{< b, i, h > |\ b \in \{\delta_1, \delta_2, +, -, 0\}, i \in I, h \in H\}$;
- $C_j = (t_j, w'_{j,0} w''_{j,0})$, $1 \leq j \leq 2$, and C_2 is the output compartment;
  - the initial multiset, $w'_{1,0} w''_{1,0}$, is given by

$$w'_{1,0} =< w_{1,0}, 1, h_1 > \cdots < w_{n_1,0}, n_1, h_{n_1} >,$$

$$w''_{1,0} =< e_1, 1, h_1 > \cdots < e_{n_1}, n_1, h_{n_1} >,$$

where $e_1 = \cdots = e_{n_1} = 0$, for all the initial multisets and initial membranes of Π. The initial multiset $w'_{2,0} w''_{2,0}$, is given by

$$w'_{2,0} = w_{i_0,0}, w''_{2,0} = e_{i_0}.$$

Initially, the indices $(I \times H)_1 = \{(1,h_1)\dots(n_1,h_{n_1})\} \setminus \{(i_0,i_0)\}$ are used in association with compartment C_1 and (i_0,i_0) for C_2. The currently used indices are $(I \times H)_c = (I \times H)_1 \cup \{(i_0,i_0)\}$.

– R'_1 and R'_2 contain the rules below.

(a.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and each rule $[u \to v]^e_h \in R$, $e \in \{+,-,0\}$, we add to R'_1 the rule $< u,i,h > \to < v,i,h > \{=< e,i,h > \wedge \overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$; these rules are applied only when the polarisation e appears in the compartment with index i and label h and none of the $< \delta_k, j, h' >$, $1 \le k \le 2$, $j \in I$, $h' \in H$, or γ_1, γ_2 appears, i.e., no dissolution or division has started and no communication with the output compartment, i_0, takes place – see below.

(a.2) For $(i,h) = (i_0,i_0)$, we add to R'_1 the rule $< u,i_0,i_0 > \to < v,i_0,i_0 > \{=< e,i_0,i_0 > \wedge \overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$ and the rule $u \to v \{= e \wedge \overline{= \gamma_{all}}\}$ to R'_2.

(b.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and $p(i,h) \neq (i_0,i_0)$, and each rule $u[\ ]^{e_1}_h \to [v]^{e_2}_h \in R$, $e_1,e_2 \in \{+,-,0\}$, we add to R'_1 the rule $< u,p(i,h) >< e_1,i,h > \to < v,i,h >< e_2,i,h > \{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$; these rules will transform $< u,p(i,h) >$ corresponding to u from the parent compartment to $< v,i,h >$ corresponding to v from the compartment with label h and index i; the polarisation is changed; as there is only one object $< e_1,i,h >$, it follows that only one single rule corresponding to the compartment can be applied at any moment of the computation.

(b.2)When $(i,h) = (i_0,i_0)$, then the rules added to R'_1 are $< u,p(i_0,i_0) >< e_1,i_0,i_0 > \to < v,i_0,i_0 >< e_2,i_0,i_0 > (ve'_2\gamma_1,2)\gamma_1 \{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$ and $\gamma_1 \to \lambda$; and the rules added to R'_2 are $e'_2 \to e_2 \{= \gamma_1\}$ and $\gamma_1 e \to \lambda$, $e \in \{+,-,0\}$. The first rule apart from simulating the communication rule, also introduces γ_1 in both compartments. In C_2 it helps changing the polarisation of it and in C_1 it helps with the synchronisation of the computation. Then the symbol disappears.

(b.3) When $p(i,h) = (i_0,i_0)$, then we add to R'_1 the rules $< u,i_0,i_0 >< e_1,i,h > \to < v,i,h >< e_2,i,h > (\gamma_2,2)\gamma_2 \{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$ and $\gamma_2 \to \lambda$. The rule $u\gamma_2 \to \lambda$ is added to R'_2. Similar to $(b.2)$, γ_2 is introduced in both compartments and in C_2 it helps removing u.

(c.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and $p(i,h) \neq (i_0,i_0)$, and each rule $[u]^{e_1}_h \to [\ ]^{e_2}_h v \in R$, $e_1,e_2 \in \{+,-,0\}$, we add the rule $< u,i,h >< e_1,i,h > \to < v,p(i,h) >< e_2,i,h > \{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$.

(c.2) When $(i,h) = (i_0,i_0)$, then we add to R'_1 the rule $< u,i_0,i_0 >< e_1,i_0,i_0 > \to < v,p(i_0,i_0) >< e_2,i_0,i_0 > (e'_2\gamma_1,2)\gamma_1 \{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$. As in $(b.2)$, we use $\gamma_1 \to \lambda$ in R'_1 and $e'_2 \to e_2 \{= \gamma_1\}$ in R'_2. We need to add to R'_2 the rule $u\gamma_1 e \to \lambda$. The rules make sure that in C_1 we simulate the communication rule and in C_2 u disappears and the polarization is changed to e_2.

(c.3) When $p(i,h) = (i_0, i_0)$, then the rule added to R'_1 is $< u,i,h >< e_1,i,h > \rightarrow < v,i_0,i_0 >< e_2,i,h > (v,2)$ $\{\overline{= \delta_{all}} \wedge \overline{\equiv \gamma_{all}}\}$. This rule simulates the communication rule and introduces v into C_2.

(d.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and $p(i,h) \neq (i_0,i_0)$, and each rule $[u]_h^e \rightarrow v \in R$, $e \in \{+,-,0\}$, we add to R'_1 the rule $< u,i,h >< e,i,h > \rightarrow < v,p(i,h) >< \delta_1,i,h >$ $\{\overline{= \delta_{all}} \wedge \overline{\equiv \gamma_{all}}\}$; all the objects corresponding to those from the compartment of index i and label h must be moved to the parent compartment - this will happen in the presence of $< \delta_1,i,h >$ when no other transformation will take place; this is obtained by using in R'_1 rules $< a,i,h > \rightarrow < a,p(i,h) >$ $\{=< \delta_1,i,h >\}$, $a \in O$ and $< \delta_1,i,h > \rightarrow \lambda$; the set $(I \times H)_c$ will change now by removing the pair (i,h) from it.

(d.2) When $p(i,h) = (i_0,i_0)$, then the rules above will become $< u,i,h >< e,i,h > \rightarrow < v,i_0,i_0 >< \delta_1,i,h > (v,2)$ $\{\overline{= \delta_{all}} \wedge \overline{\equiv \gamma_{all}}\}$ and $< a,i,h > \rightarrow < a,i_0,i_0 > (a,2)$ $\{=< \delta_1,i,h >\}$, $a \in O$.

(e) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and each rule $[u]_h^{e_1} \rightarrow [v]_h^{e_2}[w]_h^{e_3} \in R$, $e_1,e_2,e_3 \in \{+,-,0\}$; we add to R'_1 the rule $< u,i,h >< e_1,i,h > \rightarrow < v,j_1,h >< e_2,j_1,h >< w,j_2,h >< e_3,j_2,h >< \delta_2,i,h >$ $\{\overline{= \delta_{all}} \wedge \overline{\equiv \gamma_{all}}\}$ – the pair (i,h) is removed from $(I \times H)_c$ and two new pairs (j_1,h) and (j_2,h), existing in $I \times H$, with $j_1 \neq j_2$, are added to $(I \times H)_c$ and one $< u,i,h >$ is transformed into $< v,j_1,h >$ and $< w,j_2,h >$ and their associated electrical charges; then the content corresponding to compartment of index i and label h will be moved to those of index j_1 and j_2 and the same label h, hence rules $< a,i,h > \rightarrow < a,j_1,h >< a,j_2,h >$ $\{=< \delta_2,i,h >\}$, $a \in O$ are added to R'_1; finally, $< \delta_2,i,h > \rightarrow \lambda$ is also included in the set of rules of C_1; it is clear that only one division rule for the same compartment is applied in any step of the computation.

We note that in C_2 there are no rules for dissolution and division as the output compartment is not affected by these rules.

For each rule in a compartment of label h in Π there is a corresponding rule in $k\Pi$ as defined by R'_1. If the rule is in i_0 then there are rules in both R'_1 and R'_2. Every rewriting rule that is not in i_0 has a corresponding rule defined in accordance with (a.1) in R'_1; every rule in i_0 has a corresponding rule in R'_1 and one in R'_2, according to (a.2). Similarly, for communication rules in Π are defined corresponding rules in R'_1 and R'_2 – rules (b.1) to (b.3) and (c.1) to (c.3). Dissolution rules in Π have their corresponding rules in $k\Pi$ defined by (d.1) and (d.2), whereas each membrane division rule of Π has its corresponding rule in $k\Pi$ defined by (e).

The execution strategy in both compartments, C_1 and C_2, is maximal parallelism.

For a sequence of rules applied in Π, we have a corresponding sequence of rules in $k\Pi$. Obviously, for every multiset w obtained in the output compartment of Π, i_0, there is $e \in \{+,-,0\}$ and we is obtained in the output compartment of $k\Pi$, C_2.

4 Verifying kP Systems

In Section 3, we have seen the computational power of the kP systems and their ability to simulate membrane systems with active membranes. Later on in Section 6, we will illustrate how kP systems allow to model various systems that were originally specified by using different P system variants or other computational models. In addition to the study of the modelling capabilities of the kP systems, there have been investigations looking into the behaviour and properties of the models. In this respect, there have been developed methods and tools for simulating, verifying and testing the systems specified with such models. The simulation and verification aspects have been integrated into a software platform, called kPWorkbench, supporting the modelling and analysis of kP systems. The models are specified in a kP systems language, called *kP-Lingua*.

One important feature of kPWorkbench is *formal verification*. The framework supports both *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* properties by making use of the NuSMV [2] model checker. In order to facilitate the formal specification, kPWorkbench features a property language, called *kP-Queries*, comprising a list of natural language statements representing formal property patterns, from which the formal syntax of the NuSMV formulas are automatically generated. The details can be found in [12].

We illustrate the use of the model checking for a broadcasting problem introduced in [12, 4].

The broadcasting problem requires that given a tree with n nodes and height equal to m, a signal s is sent to all the nodes of the tree starting from the root. The signal will be returned back, like an acknowledgement, arriving at the root node as f, after all the nodes of the tree have been visited. The restrictive case of the broadcasting process is considered here, i.e., the signal from a node is sent to only one of its descendants, non-deterministically chosen.

In order to model the broadcasting problem we construct a kP system, $k\Pi$, with n compartments. The compartments, labelled $C_{i,j}$, are associated to the nodes of the tree as follows: $C_{1,1}$ corresponds to the root and $C_{i,j}$ corresponds to the node j from level i, $1 \leq i \leq m$, $1 \leq j \leq p_i$, where p_i is the number of nodes of level i. Formally, the kP system is defined as follows

$$k\Pi = (A, \mu, C_{1,1}, \ldots, C_{m,p_m}, C_{1,1}),$$

where $A = \{s, f, d, v\}$ is the set of objects (s the signal; f the final object arriving in the compartment $C_{1,1}$; d the object assigned initially to the compartments corresponding to internal nodes of the tree, its multiplicity equals the number of descendants of the node; and v the object appearing in the compartments that have been visited); μ contains a link between two compartments when the corresponding nodes of the tree are such that one is the child of the other one; $C_{i,j} = (t_i, w_{i,j,0})$ are the compartments, where t_i defines a type and $w_{i,j,0}$ is the initial multiset of $C_{i,j}$; $C_{1,1}$ is the output compartment.

All the compartments corresponding to the nodes from the same level i have the type t_i, $1 \leq i \leq m$. Each t_i has the form, $t_i = (R_i, \sigma_i)$. The initial multisets

are $w_{1,1,0} = sd^{p_2}$, where p_2 denotes the number of descendants of the root (i.e., the number of nodes of level 2); $w_{i,j,0} = \lambda$ if $C_{i,j}$ corresponds to a leaf node and $w_{i,j,0} = d^{p_{i,j}}$ if $C_{i,j}$ is a compartment corresponding to a node, different from the root, having $p_{i,j}$ descendants.

In the sequel for a compartment C corresponding to a node, we call descendant compartments, the compartments corresponding to the nodes that are descendants of it. Similarly, we will call C parent compartment with respect to its descendant compartments. For a compartment $C_{i,j}$, $1 < i \leq m$, $1 \leq j \leq p_i$, which corresponds to a node which is not a root, the set of rules, R_i, consists of
$r_{1,i} : sd \to (s, i+1)$ $\{ \geq d \wedge < v\}$, – unvisited, with descendants;
$r_{2,i} : s \to v(s, i-1)$ $\{ < d \wedge < v\}$, – unvisited, with no unvisited descendants;
$r_{3,i} : s \to (sd, i-1)$ $\{ < d \wedge = v\}$, – visited, with no descendants. The execution strategy is $\sigma_i = \{r_{1,i}, r_{2,i}, r_{3,i}\}$ – alternative or choice.

For the component $C_{1,1}$ corresponding to the root the set of rules R_1 contains $r_{1,1}$ derived from R_i for $i = 1$ and $r_{2,1} : s \to vf$ $\{ < d \wedge < v\}$. The execution strategy is also alternative or choice: $\sigma_1 = \{r_{1,1}, r_{2,1}\}$.

The signal s, starting from the root, goes down anytime there are unvisited descendant compartments; it will return back, level by level, after arriving to a compartment corresponding to a leaf or a node with all descendants being visited. The signal s is sent from a compartment to one of its descendants (when at least a d exists) by using $r_{1,1}$ (this will consume a d and will send s to the descendant compartment). Anytime s is in a compartment corresponding to a node having descendants and with some of them unvisited, then s will be sent to one of the descendants (rule $r_{1,i}$) consuming a d. If the compartment has been visited and receives an s then both the signal s and a d are returned back to the parent compartment (rule $r_{3,i}$); otherwise, if it receives s, but is unvisited and with no unvisited descendants then it marks it visited and returns s to the parent compartment (rule $r_{2,i}$). Finally, s returns to $C_{1,1}$ and is becomes f.

We consider the following kP system, $k\Pi_{bcast}$, with 6 compartments: $C_{1,1}$ - root, with descendants $C_{2,1}$ and $C_{2,2}$; $C_{2,2}$ has descendants $C_{3,1}$ and $C_{3,2}$; and $C_{3,2}$ has the descendant $C_{4,1}$.

We now illustrate how verification works by using query patterns specified as kP-Queries. For each property we also provide the LTL specification or the CTL specification.

The fact that the process will halt with the root node having $f = 1$, is expressed by the kP-Query pattern

eventually $(C_{1,1}.f > 0)$

and the CTL formula

```
EF C11.f > 0
```

will return true, if there *exists* an execution trace where `C11.f > 0` *eventually* holds. We know that if all children nodes are visited ($v = 1$) then parent has no children left to visit ($d = 0$). This is specified by a **Steady-state** kP-Query pattern,

$((C_{3,1}.v = 1$ **and** $C_{3,2}.v = 1)$ **implies** $C_{2,2}.d = 0)$

which is translated to the LTL formula

```
F (G ((C31.v = 1 & C32.v = 1) -> C22.d = 0)).
```

More properties are mentioned in [4].

5 Testing kP Systems

In this section we show how a kP system can be tested using automata based testing methods. We refer to the broadcasting problem discussed in Section 4. The approach is based on the method developed in [14] and [5] for cell-like P systems and applied to a kP system model of a sorting algorithm [3]. Here we follow [4].

Naturally, in order to apply an automata based testing method to a kP model, a finite automaton needs to be obtained first. In general, the computation of a kP system cannot be fully modelled by a finite automaton and so an *approximate* automaton will be sought. The problem will be addressed in two steps.

- Firstly, the computation tree of a P system will be represented as a deterministic finite automaton. In order to guarantee the finiteness of this process, an upper bound k on the length of any computation will be set and only computations of maximum k steps will be considered at a time.
- Secondly, a *minimal* model, that preserves the required behaviour, will be defined on the basis of the aforementioned computation tree.

In [4] a value of 3 has been considered for k. A finite automaton representation of the computation tree for this valus is constructed. A minimal finite cover automaton of the language defined by the previous automaton is then constructed.

In *conformance testing* one constructs a finite set of input sequences, called *test suite*, such that the implementation passes all tests in the test suite if and only if it behaves identically to the specification on any input sequence. In our case the set of input sequences is obtained from the minimal finite cover automaton. Naturally, the implementation under test can also be modelled by an unknown deterministic finite automaton. We are only interested in the behaviour of the system for sequences of length up to an upper bound k. Then, the test suite will only contain sequences of up to length k and its successful application to the implementation under test will establish that the implementation will behave identically to the specification for any sequence of length less than or equal to k. The construction of a test suite for the above system is provided in [4].

6 Applications

Many variants of P systems have been used to solved **NP**-complete problems in polynomial time [25]. We are expecting that as long as kP systems make use of rules

generating an exponential number of compartments in polynomial time (membrane division rules), they are also able to solve in a similar way such problems. Indeed, it has been shown that kP systems using membrane divison rules can solve **NP**-complete problems (3-colouring [10], subset sum and partition [9]) in polynomial time. It is also expected that these solutions are more succinct, with respect to the number of rules, than for other variants of P systems used in this respect [10].

Various other problems have been solved using kP systems, but in addition to the formal solution, a verification mechanism, based on model-checking, has been employed. Sorting problems have been considered and solutions working in linear time [7, 3] or in constant time [3], based on kP systems, have been provided. Their formal verification has been also investigated. The sorting problem, as well as a system engineering problem of modelling an e-bike system, have been considered together with a formal verification procedure, supplemented by an automata based testing approach – [4] and [19], respectively.

Applications of kP systems in systems and synthetic biology have been developed. Models of AND and OR Boolean gates [11], XOR gate [16] and quorum sensing and pulse generator [17] have been investigated together with a qualitative analysis, based on formal verification.

These kP models have been mapped into stream X-machine models [22] and high-performance simulations provided for them [1, 20, 21]. Partial implementations on a parallel hardware platform have been made [15].

7 Conclusions

Kernel P systems introduced initially as a model combining features of some existing P systems and a few other new features [6], has then become a modelling framework with computational, verification and testing capabilities [4], supported by a software platform with simulation and verification tools [12].

Future developments will include models of more complex problems, research into identifying more efficient analysis tools complementing the current modelling framework and software platform.

References

1. M. E. Bakir, S. Konur, M. Gheorghe, I. M. Niculescu, F. Ipate, High Performance Simulations of Kernel P Systems, *Proc. IEEE International Conference on High Performance Computing and Communications, 22 – 24 August, Paris* (S. Khaddaj et al., eds.), 409 – 412, 2014.
2. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV Version 2: An Open Source Tool for Symbolic Model Checking, *Proc. International Conference on Computer-Aided Verification (CAV 2002)* (W.A. Hunt et al., eds.), *Lecture Notes in Computer Science*, 2404, 359 – 364, 2002.

3. M. Gheorghe, R. Ceterchi, F. Ipate, S. Konur, Kernel P Systems Modelling, Testing and Verification – Sorting Case Study, *Proc. 17th International Conference on Membrane Computing, Milan, 25 – 29 July, 2016* (A. Leporati, C. Zandron, eds.), 161 – 174, 2016.
4. M. Gheorghe, R. Ceterchi, F. Ipate, S. Konur, Kernel P Systems: From Modelling to Verification and Testing, *Theoretical Computer Science* (accepted).
5. M. Gheorghe, F. Ipate, On Testing P Systems, *Proc. 9th Workshop on Membrane Computing* (D.W. Corne et al., eds.), *Lecture Notes in Computer Science*, 5391, 204 – 216, 2009.
6. M. Gheorghe, F. Ipate, C. Dragomir, Kernel P Systems, *Proc. 10th Brainstorming Week on Membrane Computing* (M. A. Martínez-del-Amor et al., eds.), Fénix Editora, Universidad de Sevilla, 153 – 170, 2012.
7. M. Gheorghe, F. Ipate, C. Dragomir, L. Mierlă, L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez, Kernel P Systems – Version 1, *Proc. 11th Brainstorming Week on Membrane Computing* (L. Valencia-Cabrera et al., eds.), Fénix Editora, Universidad de Sevilla, 97 – 124, 2013.
8. M. Gheorghe, F. Ipate, S. Konur, Kernel P Systems and Relationships with other Classes of P Systems, in *Multidisciplinary creativity – Homage to Gheorghe Păun on His 65th Birthday* (M. Gheorghe et al., eds.), Spandugino Publishing House, 64 – 76, 2015.
9. M. Gheorghe, F. Ipate, S. Konur, Solutions to the Subset Sum and Partition Problems Using Kernel P Systems, *Annals of Bucharest University, Computer Science*, LXII(2), 37 – 46, 2015.
10. M. Gheorghe, F. Ipate, R. Lefticaru, M.J. Pérez-Jiménez, A. Ţurcanu, L. Valencia-Cabrera, M. García-Quismondo, L. Mierlă, 3-COL Problem Modelling Using Simple kernel P Systems, *International Journal of Computer Mathematics*, 90(4), 816 – 830, 2013.
11. M. Gheorghe, S. Konur, F. Ipate, Kernel P Systems and Stochastic P Systems for Modelling and Formal Verification of Genetic Logic Gates, in *Advances in Unconventional Computing* (A. Adamatzky, ed.), Emergence, Complexity and Computation Series, Volume 1: Theory, Springer, 661 – 676, 2015.
12. M. Gheorghe, S. Konur, F. Ipate, L. Mierlă, M. E. Bakir, M. Stannett, An Integrated Model Checking Toolset for Kernel P Systems, *Proc. 16th Conference on Membrane Computing* (G. Rozenberg et al., eds.), *Lecture Notes in Computer Science*, 9504, 153 – 170, 2015.
13. M. Gheorghe, Gh. Păun, M. J. Pérez-Jiménez, G. Rozenberg, Research Frontiers of Membrane Computing: Open Problems and Research Topics, *International Journal of Foundations of Computer Science*, 24, 547 – 624, 2013.
14. F. Ipate, M. Gheorghe. Finite State Based Testing of P Systems, *Natural Computing*, 8(4), 833 – 846, 2009.
15. F. Ipate, R. Lefticaru, L. Mierlă, L. Valencia-Cabrera, H. Han, G. Zhang, C. Dragomir, M. Gheorghe, Kernel P Systems: Applications and Implementations, *Proc. 8th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA, 12 – 14 July, Huang Shan* (Z. Yin et al., eds.), 1081 – 1089, 2013.
16. S. Konur, M. Gheorghe, C. Dragomir, L. Mierlă, F. Ipate, N. Krasnogor, Conventional Verification for Unconventional Computing: a Genetic XOR Gate Example, *Fundamenta Informaticae*, 134(1-2), 97 – 110, 2014.

17. S. Konur, M. Gheorghe, C. Dragomir, L. Mierlă, F. Ipate, N. Krasnogor, Qualitative and Quantitative Analysis of Systems and Synthetic Biology Constructs using P Systems, *ACS Synthetic Biology*, 4(1), 83 – 92, 2015.
18. S. N. Krishna, M. Gheorghe, C. Dragomir, Some Classes of Generalised Communicating P Systems and Simple Kernel P Systems, *Proc. 9^{th} International Conference on Computability in Europe, 1 – 5 July, Milan* (P. Bonizzoni et al., eds.), 284 – 293, 2013.
19. R. Lefticaru, M. E. Bakir, S. Konur, M. Stannett, F. Ipate, Modelling and Validating an Engineering Application in Kernel P Systems, *18^{th} International Conference on Membrane Computing, Bradford, 24 – 28 July, 2017* (submitted).
20. R. Lefticaru, L. F. Macías-Ramos, I. M. Niculescu, L. Mierlă, Towards Agent-Based Simulation of Kernel P Systems using FLAME and FLAME GPU, *Proc. Workshop on Membrane Computing, Manchester, 11 – 15 July, 2016* (M. Gheorghe, S. Konur, eds.), Technical Report of the University of Bradford, 58 – 61, 2016.
21. R. Lefticaru, L. F. Macías-Ramos, I. M. Niculescu, L. Mierlă, Agent-Based Simulation of Kernel P Systems with Division Rules using FLAME, *Proc. 17^{th} International Conference on Membrane Computing, Milan, 25 – 29 July, 2016* (A. Leporati, C. Zandron, eds.), 195 – 216, 2016.
22. I. M. Niculescu, M. Gheorghe, F. Ipate, A. Şefănescu, From kernel P Systems to X-Machines and FLAME, *Journal of Automata, Languages and Combinatorics*, 19(1–4), 239 – 250, 2014.
23. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, 61(1), 108 – 143, 2000.
24. Gh. Păun, *Membrane Computing – An Introduction*, Springer, 2002.
25. Gh. Păun, G. Rozenberg, A. Salomaa, eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.

Space complexity of P Systems with Active Membranes: A Survey

Alberto Leporati, Luca Manzoni, Giancarlo Mauri,
Antonio E. Porreca, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy

leporati/luca.manzoni/mauri/porreca/zandron@disco.unimib.it

Summary. P systems with active membranes are a variant of P systems where membranes can be created during the computation by division of existing ones. Using this feature, one can create an exponential number of membranes in a polynomial time, and use them in parallel to solve computationally hard problems. This possibility raises many interesting questions concerning the trade–off between time and space needed to solve various classes of computational problems by means of membrane systems. In this paper we give a survey on the results on this topic.

1 Introduction

P systems with active membranes have been introduced in [8] as a variant of P systems where the membranes play an active role in the computation: an electrical charge, that can be positive (+), neutral (0), or negative (−), is associated with each membrane; the application of the rules can be controlled by means of these electrical charges. Moreover, new membranes can be created during the computation by division of existing ones. A very interesting feature of such systems is that, using these operations, one can create an exponential number of membranes in polynomial time, and use them in parallel to solve computationally hard problems.

This possibility raises many interesting questions concerning the trade–off between time and space needed to solve various classes of computational problems by means of membrane systems. In order to clarify such relations, a definition of space complexity for P systems has been proposed [10], on the basis of an hypothetical implementation of P systems by means of real biochemical materials: every single object and every single membrane requires some constant physical space.

Research on the space complexity of P systems with active membranes has shown that these devices, when using a polynomial amount of space, exactly characterize the complexity class **PSPACE**, as shown in [11] and [12]. The result has

then been generalized, showing that any Turing machine working in space $\Omega(n)$ can be simulated with a polynomial space overhead [1].

A natural research topic that follows immediately is to clarify the classes of problems solved by P systems which make use of logarithmic space. The first natural approach, when considering the use of sublinear space in the framework of membrane systems, is to compare logarithmic space P systems with Turing machines using the same amount of space. It has been shown [14] that **DLOGTIME**–uniform (a standard, weak uniformity condition for families of Boolean circuits) P systems with active membranes, using a logarithmic amount of space, are able to simulate logarithmic-space deterministic Turing machines, and thus to solve all problems in the class **L**. In [3] it is pointed out that, while logarithmic-space Turing machines can only generate a polynomial number of distinct configurations, P systems working in logarithmic space have *exponentially* many potential ones, and thus they can be exploited to solve computational problems that are harder than those in **L**. In particular, polynomial-space Turing machines can be simulated by means of P systems with active membranes using only logarithmic auxiliary space, thus obtaining a characterization of **PSPACE**.

However, an even lower amount of space suffices: P systems using only a *constant* amount of space have also been considered; in this case, it turned out [4] that, quite surprisingly, a constant amount of space is sufficient (and trivially necessary) to solve all problems in **PSPACE**. This result challenges our intuition of space, formalized in the definition of space complexity for P systems adopted so far. Thus, a more accurate estimate of the space required by a configuration of a P system was proposed. Using the new space definition, all the results involving at least a polynomial amount of space, according to the first definition, still hold. The difference appears only when P systems with severely tight bounds on the amount of space used during computations are considered.

2 Basic Notions

For a comprehensive introduction to P systems we refer the reader to *The Oxford Handbook of Membrane Computing* [9]. The definition of space complexity for P systems can be found in [10].

In order to consider general space complexity classes in the framework of P systems (i.e., including sublinear and, possibly, constant space P systems), we need to define a meaningful notion of space inspired by sublinear space definition for Turing machines: two distinct alphabets, an *INPUT* alphabet and a *WORK* alphabet, must be considered in the definition of a P system. The input objects cannot be rewritten and do not contribute to the size of the configuration of a P system. The size of a configuration is defined as the sum of the number of membranes in the current membrane structure and the total number of working objects they contain. We recall here the basic definitions related to P systems with active membranes with an input alphabet [14]:

Definition 1. *A* P system with (elementary) active membranes *having initial degree* $d \geq 1$ *is a tuple* $\Pi = (\Gamma, \Delta, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$*, where:*

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Δ *is another alphabet, disjoint from* Γ*, called the* input alphabet*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes labelled by elements of* Λ *in a one-to-one way;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are strings over* Γ *describing the initial multisets of objects placed in the* d *regions of* μ*;*
- R *is a finite set of rules over* $\Gamma \cup \Delta$*.*

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

A description of the available kinds of rule follows. This description differs from the original definition [8] only in that new input objects may not be created during the computation.

- *Object evolution rules*, of the form $[a \to w]_h^\alpha$
  They can be applied inside a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w). At most one input object $b \in \Delta$ may appear in w, and only if it also appears on the left-hand side of the rule (i.e., if $b = a$).
- *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$
  They can be applied to a membrane labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β. If $b \in \Delta$ then $a = b$ must hold.
- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β. If $b \in \Delta$ then $a = b$ must hold.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b. If $b \in \Delta$ then $a = b$ must hold.
- *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\, [c]_h^\gamma$
  They can be applied to a membrane labelled by h, having charge α, containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and charges β and γ; the object a is replaced, respectively, by b and c

while the other objects in the initial multiset are copied to both membranes. If $b \in \Delta$ (resp., $c \in \Delta$) then $a = b$ and $c \notin \Delta$ (resp., $a = c$ and $b \notin \Delta$) must hold.

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

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane several evolution rules can be applied simultaneously).
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same principle applies to each membrane that can be involved in communication, dissolution, or elementary division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence of configurations $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules of Π are applicable 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 language *recognisers* (see, e.g. [2]) by employing two distinguished objects `yes` and `no`; exactly one of these must be sent out from the outermost membrane, and only in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations

are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists. Unless otherwise specified, the P systems in this paper are to be considered confluent.

In order to solve decision problems (i.e., decide languages over an alphabet Σ), we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [6].

Definition 2. *Let $\mathcal{E}$ and $\mathcal{F}$ be classes of functions. A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* $(\mathcal{E}, \mathcal{F})$-uniform *if and only if*

- *There exists a function $f \in \mathcal{F}$ such that $f(1^n) = \Pi_n$, i.e., mapping the unary representation of each natural number to an encoding of the P system processing all inputs of length n, and defining a specific membrane as the* input membrane.
- *There exists a function $e \in \mathcal{E}$ mapping each string $x \in \Sigma^\star$ to a multiset $e(x) = w_x$ (represented as a string) over the input alphabet of Π_n, where $n = |x|$.*
- *For each $x \in \Sigma^\star$ we have $\Pi_x = \Pi_n(w_x)$, i.e., Π_x is Π_n with the multiset encoding x placed inside the input membrane.*

Definition 3. *If the mapping $x \mapsto \Pi_x$ is computed by a* single *polynomial-time Turing machine, the family $\boldsymbol{\Pi}$ is said to be $\mathcal{F}$–*semi-uniform *(where $\mathcal{F}$ is a class of functions). In this case, inputs of the same size may be associated with P systems having possibly different membrane structures and rules.*

Generally, the above mentioned classes of functions $\mathcal{E}$ and $\mathcal{F}$ are complexity classes; in the most common uniformity condition $\mathcal{E}$ and $\mathcal{F}$ denote polynomial-time computable functions, although weaker complexity classes are used for some results presented in this paper.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [6] for further details on the encoding of P systems.

Finally, we describe how space complexity for families of recogniser P systems is measured, and the related complexity classes [10, 14].

Definition 4. *Let $\mathcal{C}$ be a configuration of a recogniser P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and the total number of objects from Γ (i.e, the non-input objects) they*

contain. If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ *is a computation of* Π*, then the* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

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

The space required by Π *itself is then obtained by computing the space required by all computations of* Π *and taking the supremum:*

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

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

By $(\mathcal{E}, \mathcal{F})$-$\mathbf{MC}_{\mathcal{D}}(f(n))$ (resp. $(\mathcal{E}, \mathcal{F})$-$\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$) we denote the class of languages which can be decided by $(\mathcal{E}, \mathcal{F})$-uniform families of confluent P systems of type $\mathcal{D}$ (in the following we will mainly refer to P systems with active membranes, and we denote this by setting $\mathcal{D} = \mathcal{AM}$), where each $\Pi_x \in \boldsymbol{\Pi}$ operates within time (resp. space) bound $f(|x|)$. The corresponding class when we consider semi-uniform families is denoted by $(\mathcal{E}, \mathcal{F})$-$\mathbf{MC}^\star_{\mathcal{D}}(f(n))$ (resp. $(\mathcal{E}, \mathcal{F})$-$\mathbf{MCSPACE}^\star_{\mathcal{D}}(f(n))$).

The class of problems that can be solved in [semi-uniform] $(\mathcal{E}, \mathcal{F})$-logarithmic (respectively polynomial) space is denoted by $(\mathcal{E}, \mathcal{F})$-$\mathbf{LMCSPACE}^{[\star]}_{\mathcal{D}}$ (respectively $(\mathcal{E}, \mathcal{F})$-$\mathbf{PMCSPACE}^{[\star]}_{\mathcal{D}}$).

In [11] it has been shown that recognizer P systems with active membranes (using three polarizations) are able to solve all problems in **PSPACE** working in polynomial space and exponential time. This result shows that recognizer P systems with active membranes can solve in exponential time and polynomial space problems that cannot be solved in polynomial time and space, unless **PTIME** = **PSPACE**.

Theorem 1. $\mathbf{PSPACE} \subseteq \mathbf{PMCSPACE}_{\mathcal{D}}$.

Proof. (Sketch) The **PSPACE**-complete problem Q3SAT is solved by a P system working in polynomial space. The solution is uniform, in the sense that a fixed P system is able to solve all the instances of Q3SAT of a given size. □

In [12] it has been shown that such P systems can be simulated by Turing machines with only a polynomial increase in space requirements.

Theorem 2. $[\mathbf{N}]\mathbf{PMCSPACE}^{[\star]}_{\mathcal{D}} \subseteq \mathbf{PSPACE}$*, where* $[\mathbf{N}]$ *denotes optional non-confluence, and* $[\star]$ *optional semi-uniformity.*

Proof. (Sketch) The inclusion $\mathbf{PMCSPACE}^\star_{\mathcal{D}} \subseteq \mathbf{PSPACE}$ is proved by simulating a nondeterministic P system working in polynomial space by a Turing machine working in polynomial nondeterministic space, which can then be reduced to polynomial deterministic space by using Savitch's theorem [7]. □

Together, the previous results give a precise characterization of the class **PSPACE** in terms of space complexity classes for membrane systems.

This result was then generalized in [1], by showing that arbitrary single-tape Turing machines can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. As a consequence, the classes of problems solvable by P systems with active membranes and by Turing machines coincide up to a polynomial with respect to space complexity.

Theorem 3. *Let M be a single-tape deterministic Turing machine working in time $t(n)$ and space $s(n)$, including the space required for its input. Then there exists a uniform family of confluent P systems $\boldsymbol{\Pi}$ with restricted elementary active membranes operating in time $O\big(t(n)s(n)\log s(n)\big)$ and space $O\big(s(n)\log s(n)\big)$ such that $L(\boldsymbol{\Pi}) = L(M)$.*

Proof. (Sketch) The techniques used in [11] and [12] to simulate Turing machines via uniform families of P systems do not seem to apply when the space bound is super-exponential, because membranes are identified by binary numbers. In fact, when dealing with a super-exponential number of different membrane labels, such numbers would be made of a super-polynomial number of digits, and such systems cannot be built in a polynomial number of steps by a deterministic Turing machine, as required by the notion of polynomial-time uniformity usually employed.

Instead, multiple copies of a single "dot" object are used to represent the cell numbers in unary notation, and all membranes representing cells of the Turing machine have the same label. A configuration of M where q is the state of the machine, the visited portion of the tape has length m, the string on the visited portion of the tape is $w = w_1 \dots w_m \in \Sigma_m$, and the head is placed on tape cell $p \in \{1, \dots, m\}$, is encoded as it follows:

- Three membranes labelled by q, p, and m contain, respectively, the unary encoding of q, p, and m, that is, as many copies of the dot object as the corresponding value;
- The i–th cell of the Turing machine M containing the j–th symbol of the alphabet, is simulated by means of a membrane containing the value $K \times i + j$ in unary notation.

To simulate a computation step of the Turing machine M we first need to identify, among all membranes labelled by t, the one corresponding to the cell located under the tape head of M. Notice that these membranes are externally indistinguishable, and they differ only in the unary value contained in it. A nondeterministically guess among all these membranes is performed, which is then checked to verify if it is indeed the right one; if this is not the case, then the membrane is marked, and the process repeated until we eventually find the correct one.

Once correctly identified the involved membrane, the computation step is simulated, working on numbers in unary notations through a subroutine that simulate a register machine to update the configuration of the P system, according to the transition step of the Turing machine. □

From Theorem 3 we obtain inclusions of complexity classes for Turing machines and P systems when the space bound is at least linear (since we are dealing with single-tape Turing machines). In particular, for every function $f(n) \in \Omega(n)$ the following inclusions hold: $\mathbf{TIME}(f(n)) \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MC}_{\mathcal{AM}}(O(f(n)^3))$ and $\mathbf{SPACE}(f(n)) \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(f(n)^2))$.

Moreover, by combining the previous results, we can prove *equality* between space complexity classes for P systems and Turing machines under some (not very restrictive) assumptions on the set of space bounds we are interested in.

Theorem 4. *Let $\mathcal{F}$ be a class of functions $\mathbb{N} \to \mathbb{N}$ such that*

- *$\mathcal{F}$ contains the identity function $n \mapsto n$;*
- *If $s(n) \in \mathcal{F}$ and $p(n)$ is a polynomial, then there exists some $f(n) \in \mathcal{F}$ with $f(n) \in \Omega(p(s(n)))$.*

Then $\mathbf{SPACE}(\mathcal{F}) = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(\mathcal{F})$. In particular, we have the following equalities:

$$\mathbf{PSPACE} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{AM}}$$
$$\mathbf{EXPSPACE} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{EXPMCSPACE}_{\mathcal{AM}}$$
$$2\mathbf{EXPSPACE} = (\mathbf{L}, \mathbf{L})\text{-}2\mathbf{EXPMCSPACE}_{\mathcal{AM}}$$
$$k\mathbf{EXPSPACE} = (\mathbf{L}, \mathbf{L})\text{-}k\mathbf{EXPMCSPACE}_{\mathcal{AM}}.$$

Another consequence of the possibility of P systems to simulate Turing machines with a polynomial overhead and vice versa is that we can translate theorems about the space complexity of Turing machines into theorems about P systems. As an example, the Savitch's theorem and the Sapce hierarchy theorem for Turing machines can be proved almost immediately for large enough space complexity bounds.

3 Simulating Logarithmic–Space Turing Machines

To consider membrane systems working in logarithmic space is one of the first natural research topic that has been addressed once obtained the result described in the previous section. We first recall a result from [14] showing that P systems with active membranes, using a logarithmic amount of space, are able to simulate logarithmic-space deterministic Turing machines, and thus to solve all problems in the class **L**.

In order to consider such systems, we need to define a uniformity condition for the families of P systems that is weaker than the usual **P** uniformity, to avoid the possibility to solve a problem directly by using the Turing machine that builds the P systems we use to compute. One such possibility is to consider **DLOGTIME**-uniformity, defined on the basis of **DLOGTIME** Turing machines [5]. Another problem that the efficient simulation of logarithmic space Turing machines (or

other equivalent models) has to face, is that it cannot use a polynomial number of working objects, to avoid violating the logarithmic space condition.

It has been shown in [14] that such problems can be avoided by a simulation that uses membrane polarization both to communicate objects through membranes as well as to store some information.

Theorem 5. *Consider a deterministic Turing machine M, having an input tape of length n, and with a work tape of length $O(\log n)$. Then, there exists a* (**DLOGTIME**, **DLOGTIME**)*-uniform family $\boldsymbol{\Pi}$ of confluent recogniser P systems with active membranes that works in logarithmic space such that $L(M) = L(\boldsymbol{\Pi})$.*

Proof. (sketch) Consider a Turing machine M working in logarithmic space. The P system Π_n that simulates M on input of length n is composed of:

- A skin membrane containing a *state object* object $q_{i,w}$ to indicate that M is currently in state q and its tape heads are on the i-th and w-th symbols of the input and work tape, respectively.
- $O(\log n)$ nested membranes (INPUT tape membranes) containing, in the innermost one, the input symbols of M, and $O(\log(n))$ membranes to store the work tape of M (WORK tape membranes).
- Two sets of membranes, whose size depend on the dimensions of the input and the working alphabet of M (SYMBOL membranes), respectively.

To simulate a computation step of M, the state object enters the INPUT membranes, storing the bits corresponding to the actual position of the INPUT head of M in their polarizations. Only one object (corresponding to the INPUT symbol actually read) can travel to the outermost membrane by using send-out rules; the other objects stop moving because they have the wrong charges. Then, the state object identifies the symbol actually under the WORK head (using the WORK tape membranes) and proceeds to simulate the transition of M using the SYMBOLS membranes.

Each P system Π_x (simulating each $M(x)$ such that $|x| = n$) only requires $O(\log |x|)$ membranes and objects besides the input objects; moreover, the family $\boldsymbol{\Pi}$ is (**DLOGTIME**, **DLOGTIME**)-uniform. The time required by the simulation is $O\big(n \cdot t(n)\big)$, where $t(n)$ is the maximum number of steps performed by M on inputs of length n. □

An immediate corollary of Theorem 5 is that the class of problems solved by logarithmic-space Turing machines is contained in the class of problems solved by (**DLOGTIME**, **DLOGTIME**)-uniform, logarithmic-space P systems with active membranes.

Corollary 1. $\mathbf{L} \subseteq$ (**DLOGTIME**, **DLOGTIME**)-$\mathbf{LMCSPACE}_{\mathcal{AM}}$. □

4 Simulating Polynomial-Space Turing Machines in Logarithmic Space

The result presented in the previous section only represents a lower bound for the power of logarithmic-space P systems; as a matter of fact, already in [14] it was conjectured that it could be improved, as P systems working in logarithmic space have an *exponential* number of different configurations, which could possibly be used to efficiently solve harder problems than those in the class **L**. It turned out [3] that this is the case, and that polynomial-space deterministic Turing machines can be simulated by means of P systems with active membranes using only logarithmic auxiliary space, thus characterising **PSPACE**.

The simulation was based on two key ideas. First, input objects (of the form τ_i) are distributed, during the computation, in various substructures. Apart from an initial phase, the value of τ is disregarded: the symbol σ written on the i-th tape cell of the Turing machine being simulated can be inferred from the label of the substructure that contains τ_i. The second idea is applied when querying the symbol under the tape head: the position i of the head is written in binary in the electrical charges of the membranes composing the substructure where the object τ_i is placed, so that the only input object having the correct subscript can leave the substructure corresponding to the sought symbol, and reach the skin membrane. The depth of each substructure is logarithmic, thus allowing to represent a polynomial number of possible head positions. As a result, we can simulate any polynomial space computation of a deterministic Turing machine with only a logarithmic number of symbols (plus a polynomial number of read-only input symbols) and membranes.

Theorem 6. *Let M be a single-tape deterministic Turing machine working in polynomial space $s(n)$ and time $t(n)$. Then, there exists an $(\mathbf{L}, \mathbf{L})$-uniform family $\boldsymbol{\Pi}$ of P systems with active membranes using object evolution and communication rules that simulates M in space $O(\log n)$ and time $O(t(n)s(n))$.*

Proof. (sketch) Let $x \in \Sigma^n$ be an input string, and let $m = \lceil \log s(n) \rceil$ be the minimum number of bits needed in order to represent the tape cell indices $0, \ldots, s(n)-1$ in binary notation. The P system Π_n, associated with the input length n, has a membrane structure consisting of an external skin membrane that contains, for each symbol of the tape alphabet of M, the following set of membranes, linearly nested and listed from the outside in:

- a *symbol-membrane*;
- a *query-membrane*;
- for each $j \in \{0, \ldots, m-1\}$, a membrane labelled by j_σ.

An arbitrary configuration of M on input x is encoded by a configuration of Π_x as follows:

- the outermost membrane contains the *state-object* q_i, (where q is the current state of M, and i is the current tape head position);

- if membrane $(m-1)_\sigma$ contains the input object τ_i, then the i-th tape cell of M contains the symbol σ.

The symbol written on the i-th tape cell of M can be inferred from the label of the substructure which contains the corresponding input symbol τ_i. Notice that a logarithmic depth membrane structure allows to represent a polynomial number of possible head positions.

The state-object q_i queries each membrane substructure, by encoding in binary the tape position i on the electrical charges of the membranes. Only the symbol whose subscript is i can reach the skin membrane and be used to conclude the simulation of a computation step.

The family $\boldsymbol{\Pi}$ described above is $(\mathbf{L}, \mathbf{L})$-uniform, and each P system Π_x uses only a logarithmic number of membranes and a constant number of objects per configuration, besides the input objects, which are never rewritten. Π_x works in space $O(\log n)$ and in time $O(t(n)s(n))$. □

As a consequence, we have the following:

Theorem 7. *For each class $\mathcal{D} \subseteq \mathcal{AM}$ of P systems with active membranes using object evolution and communication among their rules we have*

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}} = \mathbf{PSPACE}.$$

Proof. The inclusion $\mathbf{PSPACE} \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}}$ follows immediately from Theorem 6. By definition, the class $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}}$ is included in the class $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}}$. Finally, to prove the inclusion of $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}}$ in **PSPACE** it suffices to simulate P systems by means of Turing machines, which can be carried out with just a polynomial space overhead, as shown in [11, 1]. □

This was the first case where the space complexity of P systems and that of Turing machines differ by an exponential amount. Since, as previously said, **PSPACE** had already been proved to be characterised by *polynomial*-space P systems, these results also highlight a gap in the hierarchy of space complexity classes for P systems: super-polynomial space is required in order to exceed the computational power of logarithmic space.

5 Constant–Space P systems

After considering P systems with active membranes working in logarithmic space, a natural question arises concerning the power of such systems using only a constant amount of space. Surprisingly it turned out that constant space is sufficient to simulate polynomial-space bounded deterministic Turing machines, as proved in [4]:

Theorem 8. $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(1)) = \mathbf{PSPACE}$.

Proof. (sketch) Let $L \in \mathbf{PSPACE}$, and let M be a Turing machine deciding L in space $p(n)$. We can construct a family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ such that $L(\boldsymbol{\Pi}) = L$ by letting $F(1^n) = \Pi_n$, where Π_n is the P system simulating M on inputs of length n, and

$$E(x_0 \cdots x_{n-1}) = x_{1,1} \cdots x_{n-1,n-1} \sqcup_n \cdots \sqcup_{p(n)-1},$$

i.e. by padding the input string x with $p(n) - n$ blank symbols $\sqcup$ before indexing the result with the positions of the symbols on the tape.

The simulation relies on two main ideas. As in the previous proof of Theorem 6, input objects of the form τ_i are distributed in substructures, and the symbol written on the i-th tape cell of M can be inferred from the label of the substructure where the corresponding input symbol τ_i is placed. The second idea is that it is possible to "read" a subscript of an input object τ_i without rewriting it and by using only a constant number of additional objects and membranes: in particular, a timer object is used to change the charge of a membrane after a requested amount of steps. Any other object that was counting together with the timer is able to observe the charge of the membrane, and thus obtain the designed value.

Since at each computation step only a constant number of working objects and membranes are present, then the simulation requires, according to Definition 4, a constant amount of space. Moreover, both F and E can be computed in logarithmic space by Turing machines, since they only require adding subscripts having a logarithmic number of bits to rules or strings having a fixed structure, and the membrane structure is fixed for all Π_n. This proves the inclusion of **PSPACE** in $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$, while the reverse inclusion is proved in [11]. □

6 Rethinking the Definition of Space

The result of Theorem 8 shows that all problems in **PSPACE** can be solved by constant-space P systems with active membranes. This rises some natural questions about the definition of space complexity for P systems adopted until now [10]. Does counting each non-input object and each membrane as unitary space really capture an intuitive notion of the amount of space used by a P system during a computation? Is it fair to allow a polynomial padding of the input string when encoding it as a multiset?

In [4], it was highlighted that the constant number of non-input objects appearing in each configuration of the simulation actually encode $\Theta(\log n)$ bits of information, since they are taken from an alphabet Γ of polynomial size. According to the original definition of space recalled in Section 2, each of these objects would only require unitary space, whereas the binary representation of the subscript i requires $\log p(n) = \Theta(\log n)$ bits. It may be argued that this amount of information needs a proportional amount of physical storage space. Similarly, each membrane label contains $\Theta(\log |\Lambda|)$ bits of information, which must also have a physical counterpart.

The information stored in the *positions* of the objects within the membrane structure is also not taken into account by Definition 4. However, the information on the location of the objects is part of the system and it is *not* stored elsewhere, exactly as the information on the location of the tape head in a Turing machine, which is not counted as space.

Due to the above considerations, in [4] an alternative definition of space was proposed:

Definition 5. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the number of membranes in the current membrane structure* multiplied by $\log |\Lambda|$*, plus the total number of objects from Γ (i.e., the non-input objects) they contain* multiplied by $\log |\Gamma|$.

Adopting this stricter definition does not significantly change space complexity results involving polynomial or larger upper bounds, i.e., the complexity classes $\mathbf{PMCSPACE}_{\mathcal{AM}}$, $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$, and larger ones remain unchanged.

As for padding the input string, one may argue that this operation provides the P system with some "free" storage, since input objects are not counted by Definition 4. The proof of Theorem 8 exploits the ability to encode an input string of length n as a polynomially larger multiset in a substantial way, as allowed by the most common uniformity conditions, including **P** and **LOGSPACE**-uniformity, but also weaker ones such as $\mathbf{AC}^0$ or **DLOGTIME**-uniformity.

The simulation described in the previous section would require logarithmic space according to Definition 5. Also the space bounds of the simulation of polynomial-space Turing machines by means of logarithmic-space P systems with active membranes described in Section 4 also increase to $\Theta(\log n \log \log n)$, since in that case each configuration of the P systems contains $\Theta(\log n)$ membranes with distinct labels and $O(1)$ non-input objects. Both simulations would be limited to *linear*-space Turing machines, rather than polynomial-space ones, if input padding were disallowed.

7 Final Remarks

In this paper we survey recent results concerning complexity of P systems with active membranes. The results showed that such P systems can be simulated by Turing machines with only a polynomial increase in space requirements and, moreover, that arbitrary single-tape Turing machines can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. This leads to prove equalities among space complexity classes for P systems and Turing machines (as long as the sets of space bounds satisfies some reasonable properties). In particular, complexity classes defined in terms of polynomial, exponential, double exponential, ..., n-fold exponential space coincide for the two kinds of device.

It has also been shown that the class **PSPACE** can be characterized by P systems with active membranes using logarithmic space or even constant amount of space. In view of the last result, a new definition of space for P systems has been proposed, that also take into account the number of bits necessary to encode the non-input objects and the labels of the membranes. While the new definition does not change any result involving an amount of space which is polynomial or larger, it changes the result for sublinear space. In particular, according to the new definition the simulation used to prove that constant space P systems with active membranes characterize **PSPACE** would now require logarithmic space.

References

1. A. Alhazov, A. Leporati, G. Mauri, A.E. Porreca, C. Zandron, Space complexity equivalence of P systems with active membranes and Turing machines, Theoretical Computer Science, 529, 2014, 69–81.
2. E. Csuhaj-Varju, M. Oswald, Gy. Vaszil, P automata, Handbook of Membrane Computing. Gh. Păun et al. (Eds.), Oxford University Press, 2010, 144-167.
3. A. Leporati, G. Mauri, A.E. Porreca, C. Zandron, A gap in the space hierarchy of P systems with active membranes, Journal of Automata, Languages and Combinatorics 19 (2014) 14, 173184.
4. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Constant–space P systems with Active Membranes, Fundamenta Informaticae, to appear.
5. D.A. Mix Barrington, N. Immerman, H. Straubing, On uniformity within NC^1. Journal of Computer and System Sciences 41(3), 1990, 274–306.
6. N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Natural Computing 10(1), 2011, 613–632.
7. C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1993.
8. Gh. Păun, P systems with active membranes: Attacking NP-complete problems, J. of Automata, Languages and Combinatorics 6(1), 2001, 75–90.
9. Gh. Păun, G. Rozenberg and A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, 2010.
10. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, Introducing a space complexity measure for P systems, Int. J. of Comp., Comm. & Control 4(3), 2009, 301–310.
11. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems with active membranes: Trading time for space, Natural Computing 10(1), 2011, 167–182.
12. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems with active membranes working in polynomial space, Int. J. Found. Comp. Sc., 22(1), 2011, 65–73.
13. A.E. Porreca, G. Mauri, C. Zandron, Complexity classes for membrane systems, RAIRO-Theor. Inform. and Applic. 40(2), 2006, 141-162.
14. A.E. Porreca, C. Zandron, A. Leporati, G. Mauri, Sublinear space P systems with active membranes, Membrane Computing: 13th International Conference, LNCS, CMC 2012, Springer, Berlin, 2013, 342-357.

A Survey of Parallel Simulation of P Systems with GPUs

Miguel A. Martínez-del-Amor, Agustín Riscos-Núñez, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Dept. Computer Science and Artificial Intelligence, University of Seville
Avda. Reina Mercedes S/N, 41012, Sevilla, Spain
{mdelamor,ariscosn,marper}@us.es

Summary. P system simulators become essential for model verification and validation, since they reproduce the semantics of the models in an automatic way. For this reason, in the literature, many authors have proposed several simulation tools. However, in order to handle large instances in an efficient way, parallel simulators come into play.

High Performance Computing is a research branch that brings efficient tools for scientific purposes. For decades, many parallel platforms and architectures have been designed, with the goal of accelerating compute-demanding applications. But it was 10 years ago, that this field was revolutionized with the dawn of GPU computing through CUDA. This technology allowed programmers to run general-purpose parallel code in GPUs, harnessing in a simplified manner the large amount of processors within a GPU.

Many authors have chosen this technology for accelerating the simulation of their P system models. Recently, this topic has captured the attention of more researchers. Therefore, in this paper we survey the related work on GPU-based simulators for P systems, and its evolution over the time until today.

Key words: Membrane Computing, P systems, Parallel Computing, GPU computing, CUDA

1 Introduction

Simulating P systems [27] is a task that has become important in the past years [28]. Indeed, validating a model requires an automatic procedure in order to make it feasible. Specifically, when employing P systems as a modeling framework for biological phenomena in *Computational Systems Biology* or *Population Dynamics* [12], simulation tools are critical since they enable experimental validation and virtual experimentation.

Today's challenge in P system simulators is their accuracy and efficiency [19]. For the former, the solution is to define simulation algorithms that represent in a more reliable way the semantics of the corresponding P system model. For example, non-determinism, rule competition, stochastic/probabilistic execution of rules

are very difficult to handle in a simulator. For the latter, parallel platforms help to accelerate the execution, so both the simulation algorithms and their implementations must be adapted for the specific kind of parallelism.

A new trend in High Performance Computing is to use heterogeneous systems, where commodity CPUs have attached a massively parallel co-processor named *accelerator*. This is the case of GPU computing, which became popular with the introduction of CUDA 10 years ago [30]. Although the GPU is the device in charge of rendering the graphics in a computer, its evolution has given a highly parallel processor, with thousand of lightweight cores, that can be harnessed for scientific - general purpose - computing [15]. Using CUDA or OpenCL, programmers design their code for an abstract parallel architecture, where *threads* are executed in a SIMD fashion, and distributed into blocks.

Since GPUs offer a shared memory system with a high degree of parallelism, they have been considered for accelerating the simulation of some P system models [20]. One example is the *PMCGPU* (Parallel simulators for Membrane Computing on the GPU) [33] project, where the authors published the source code of the first simulators of this kind.

In this paper, we provide an updated survey of the developed P systems simulators on the GPU. Moreover, we discuss some strategies followed, this article serving as a tutorial as well. The paper is structured as follows: Section 2 introduces the main concepts of GPU computing, while Section 3 gives an overview of the parallel simulators of some P system models. Finally, Section 4 provides some conclusions and future research lines.

2 GPU computing

The *GPU* (Graphics Processor Unit) is in the core of graphics cards [16]. They were first conceived for rendering and computing color attributes of pixels in a parallel way. With the fast growth of the graphics market in the recent years, the GPU have evolved into a parallel processor with a special nature. The cores found in a GPU are much simpler than normal CPUs, but in a larger number. They provide a system optimized for data parallelism, where threads are executed in SIMD (Single Instruction Multiple Data).

After the introduction of *CUDA* (Compute Unified Device Architecture) [31, 16] by NVIDIA, GPUs offered a programming model that abstracts the GPU architecture to programmers. Hence, it is enough to learn some extensions to C/C++ language (CUDA extensions) and the programming model, while the CUDA driver executes the actual code on the GPU. The programming is flexible, but the achieved performance depends on how the implementation fits data parallelism.

On the other side, *OpenCL* (Open Compute Language) [32] were introduced in order to enable the usage of any kind of parallel devices using a similar abstracted architecture as CUDA [16]. In fact, any modern GPU supports the execution of

OpenCL regardless the brand (NVIDIA, AMD, Intel...). Although today many brands have almost abandoned the support of OpenCL, it is still best choice for those GPUs not supporting CUDA. The concepts in OpenCL and CUDA are similar, so in what follows we will only focus on the description of the latter.

2.1 CUDA programming model

In the CUDA programming model, the CPU (*host*) takes control of the execution flow, and permits the GPU (*device*) to execute a piece of code (*kernel* function) in parallel. The execution is carried out by a *grid* of *threads*. Typically, a grid is composed of thousands of threads, what allows to increase the occupancy of the hardware resources. This is required in order to hide stalls in the execution of threads (given by dependencies, memory accesses, etc.). The grid is a two-level hierarchy (see Figure 1), where *threads* are arranged into *thread blocks*. All blocks have the same number and organization of threads. Each block is identified by a two dimensional identifier, and each thread within its block by a three dimensional identifier (ID). In this way, any thread can be identified by the combination of both thread and thread block identifiers. The execution of threads inside a block can be synchronized by *barrier* operations (`__syncthreads()`), and threads of different blocks can be synchronized only by finishing the execution of the kernel.

Another aspect in CUDA is that the memory hierarchy is explicitly managed. This is composed in several levels, each one offering different speeds and storage properties. *Global memory* is the largest but the slowest memory in the system. It is accessed by the host (where the input and output data are allocated) and by any thread in execution. An *L2 cache* memory system is built in recent GPUs, allowing to speedup the access to global memory in a transparent way (this L2 cache system is hidden to programmers). *Shared memory* is the smallest but fastest memory. It is accessed by threads belonging to the same block. Normally, performance of CUDA applications depends on how much shared memory is exploited. Finally, every thread has access to its own variables in a very fast way allocated in *registers*. Thus, the most efficient way to structure an algorithm is as follows: (1) threads of each block read their corresponding data portion from global memory to shared memory, (2) threads work with the data directly on the shared memory, and (3) threads copy these data back to global memory.

A well-known strategy in parallel programming, and used also in CUDA, is *tiling*. This strategy seeks to combine the previous structure of three phases with partitioning data, so that the three phases are repeated for each data portion (or *tile*), minimizing accesses to global memory. Finally, it is worth to remark that the access to both shared and global memory achieves the best performance when threads read contiguous portions of data (e.g., adjacent positions of an array). This is called *coalesced memory access*, and maximizes the memory bandwidth utilization.

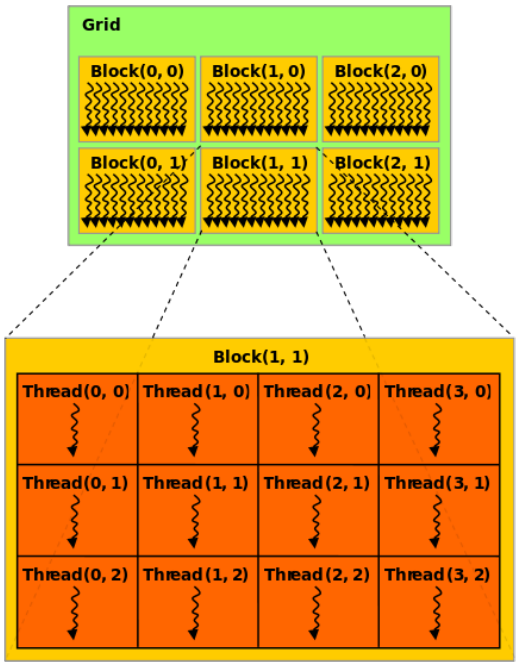


Fig. 1. Threading organization in CUDA. Threads are executed in a grid, and they are organized in blocks. From [31].

2.2 GPU architecture

As mentioned above, the GPU architecture has evolved since its first introduction, offering even more computing capabilities. In general, it consists of a set of *Streaming Multiprocessors (SMs)* containing *Streaming Processors (SPs, or cores)*. The number of them depends on the GPU: every microarchitecture version has a given amount of SPs per SM, and the number of SMs depends on the device range. SMs are based on the *SIMT (Single-Instruction Multiple-Thread)* model, where all the threads execute the same instruction on different piece of data. SMs create, manage, schedule and execute threads in groups of 32 threads (which is the branching granularity of NVIDIA GPUs), called *warp*. Individual threads of the same warp must start together at the same program address, enforcing a SIMD execution. However, they are free to branch and execute independently, but at cost of serialization and performance.

2.3 GPU features

Today, the GPU is considered as a relatively low cost technology offering a high level of parallelism, but at expenses of dedicated designs to maximize the GPU

utilization, requiring expertise knowledge to achieve best performance. This can be observed from the large amount of research and theses available in the literature.

In summary, we can conclude that a GPU leverages:

- *Good performance*: for example, the NVIDIA Tesla K40 delivers 1.43 TeraFLOPS double-precision peak floating point performance, 4.29 TeraFLOPS of single-precision, and 288 GBytes/s of global memory bandwidth;
- *An efficiently synchronized platform*: GPU implements a shared memory system, avoiding communication overload;
- *A medium scalability degree*: the amount of resources depend on the GPU model, e.g., a K40 includes 2880 cores and 12 GBytes of memory. If the resources of a GPU is not enough, there are more scalable solutions such as multi-GPU systems, but they then require communication among nodes;
- *Low-medium flexibility*: although CUDA programming is based on C++, and hence programmers are free to use the same data structures as in CPU, both the algorithm and the data structure have to be adapted for best performance on GPUs.

3 P systems simulators on GPUs

As discussed above, P system simulations can be accelerated by taking advantage of today solutions in High Performance Computing. Current cutting-edge parallel technologies offer enablers for many scientific applications. One example is the importance of GPU computing, which is a requirement today to run deep convolutional neural networks [30].

Specifically, real ecosystems models based on P systems are very time demanding when being simulated. The need of running accurate, but expensive, simulation algorithms several times in order to collect statistics points out to look for accelerators such as GPUs. In this way, model designers and expert users can interact with their models close to real time.

For this reason, the project *PMCGPU* (Parallel simulators for Membrane Computing on the GPU) [33] was initiated. Here, simulators for P systems on GPUs are available in open source. In the forthcoming subsections we will go over the simulators developed for GPUs, paying attention to their main features and some design topics.

3.1 P systems with active membranes

The first test of concept for simulating P systems on GPUs was applied for P systems with active membranes using CUDA (Cecilia et al. [7]). The aim was to perform just one computation in order to avoid non-determinism, so confluence is a required property for the simulated P systems. Moreover, this requirement is used for another design decision: to select the "lowest-cost computation path" for

the simulator; that is, the one in which least membranes and communication are required.

The simulation algorithm consists basically in a loop over the transition steps, reproducing one computation path of the whole tree. For each transition, two stages are applied: *selection* and *execution*. Selection of rules is the most time-consuming stage, since it implements all the semantics of the model concerning how rules are applied. In this way, rules are chosen for a given P system configuration, together with the number of times to apply each one. Preferences to rules that lead to least membranes (e.g. dissolution over division rules) are imposed. The result of this stage is used for the next one, which is the execution of the rules; that is, updating the P system configuration. This two-staged strategy allows to synchronize the application of rules within and among membranes.

The parallelism of P systems is mapped on the GPU using the double-parallel nature of both of them: (elementary) membranes are assigned to thread blocks, and a subset of rules to threads. For the latter, in fact, each thread was in charge of selecting rules for a portion of the defined objects in the alphabet. Note that for selection, looking to objects or rules needs similar strategies, since in active membranes, rules have no cooperation.

It is easy to note that this mapping of parallelism is naive: the CUDA simulator assumes by default that all the defined objects in the alphabet will be present within each membrane, allocating memory space and assigning resources (threads) for all of them. Although this is the worst case, it does not take place in the majority of P systems to be simulated. Thus, the performance of the simulator completely depends on the simulated P system, and drops as long as the variety of different objects appearing in membranes decreases.

The performance of the simulator was tested on an NVIDIA Tesla C1060 GPU (240 cores, 4GB memory) by using two benchmarks [19]: a simple test P system designed to stress the simulator (A), and a family of P systems solving the SAT problem (B). Up to 7x of speedup was reported for (A), and 1.67x for (B). Using these results, three indicators that affect performance were identified [19, 20]: density of objects per membrane, rule intensity and communication among membranes.

The simulator is available in the PMCGPU project as a standalone tool. It receives a file in binary format, which specifies the P system to simulate and the initial configuration. The output is then given in text format for debugging purposes.

Maroosi et al. [26] improved this simulator by 38x, taking advantage of shared memory and minimizing data transfers. A dependency graph is constructed from the set of rules of the input model, so that rules having common objects in the left-hand side and in the right-hand side (respectively) are more likely to be grouped in a node of the graph. This is then employed to distribute those rules that will trigger others in the same thread block, what allows to reduce the communication between them. To date, the authors are extending the simulators in order to allow the simulation of more than two levels in the membrane structure.

3.2 A family solving SAT with active membranes

A subsequent project after simulating P system with active membranes was to focus on a specific family of this model solving SAT in linear time (Cecilia et al. [8, 9]). By analyzing this specific solution, the authors were able to put an upper bound to the number of existing objects in membranes, maximizing in this way the work done by threads. Moreover, the simulation algorithm is based on the stages identified in the computation of any P system in the family: generation, synchronization, check-out and output.

The design of the CUDA simulator follows the same basis as its predecessor: a thread block is assigned to each elementary membrane (which in turn encodes a truth assignment to the CNF formula), and each thread to each object that might appear in the membranes. As mentioned above, the number of objects to be represented in the data structures is decreased. In this case, it is enough to store only the objects appearing in the input multiset (which is a literal of the CNF formula). Therefore, threads are assigned to each object of the input multiset.

A hybrid simulator was also proposed, with the aim at improving the usage of resources, and so, achieving better speedups. In this hybrid solution, the execution of the first stages are reproduced exactly as the P system model does, but the last ones are not. Although they give the same result, they are more efficiently executed on the GPU.

The experiments carried out on an NVIDIA Tesla C1060 GPU report up to 63x of speedup for the CUDA simulator against sequential solution, and the hybrid CUDA simulator outperforms the CUDA simulator in 9.3x.

These simulators receive as input a DIMACS CNF file codifying an instance of SAT through a CNF formula. The output is a summary of the codification, and the answer `yes` or `no`. Therefore, they merely behave as a SAT solver whose design is based on a P systems based solution.

Further developments were carried out to this family of simulators, with the aim at being better tailored to the GPU idiosyncrasy. More efficient strategies for GPUs were performed, such as tailing, as well as newer GPU architectures, multi-GPU systems and supercomputers were utilized [9, 10]. An improvement of 60-90% was achieved by using these specific optimizations to the CUDA code.

3.3 A family solving SAT with tissue-like P systems

In order to explore which P system ingredients are better suited to be handled by GPUs, another solution to SAT based on a family of tissue P systems with cell division was simulated (Martínez-del-Amor et al. [23]). The simulation algorithm is based on the 5 stages identified in the computation of the P systems in the family: generation, exchange, synchronization, check-in, and output.

The CUDA simulator design is similar to the one used in [8]. Each thread block is assigned to each cell. However, the number of objects to be placed inside each cell in the memory representation is increased, respecting to the solution in active membranes. On the contrary, this simulator does not need to store nor handle

charges associated to membranes. Threads are then used differently in each stage, maximizing their usage in each case.

Experiments on an NVIDIA Tesla C1060 GPU showed that the CUDA simulator outperforms a sequential version by 10x. This demonstrated that the usage of charges associated to membranes helped to save instantiation of objects, while they represent a lightweight ingredient to be represented and processed by threads.

3.4 Population Dynamics P systems

Population Dynamics P (PDP) systems define a formal modeling framework for real ecosystems [11]. Their efficient simulation is critical for virtual experimentation and experimental validation, as discussed above. PDP systems are multienvironment models, and typically, simulation tools need to run several simulations of them since the execution of rules follows a probabilistic distribution.

The selected simulation algorithm was the DCBA [25], since it provided better accuracy in the simulation results. DCBA consists of two major stages, selection and execution. Selection is in turn sub-divided into three micro-stages: phase 1 (distribution of objects), phase 2 (maximality) and phase 3 (probability). The main component of this algorithm is a distribution table, employed to distribute the objects in a proportional way between competing rules, i.e., with overlapping left-hand sides. In turn, rules are grouped into *rule blocks*, when having the same left-hand side. The main problem arises when simulating large PDP system models, since the table can be too large and sparse. Therefore, the simulator workarounds the table and, instead, works directly with the information of rules.

A first approach was to parallelize the simulator with OpenMP for multi-core CPUs. This was done in three different ways [21]: 1) by simulations, 2) by environments and 3) a hybrid approach. The experiments were ran on two multi-core processors: the Intel i5 Nehalem and i7 Sandy Bridge, achieving speedups of up to 2.5x by using all the cores of a single socket 4-core Intel i7. Experiments also indicate the simulations are memory-bandwidth bound and the portion of the code we parallelized consumes over 98% of the runtime in serial. From this initial work, the authors concluded that parallelizing by simulations or hybrid techniques yields the largest speedups.

The second approach was the creation of a CUDA simulator for PDP systems [24]. The design of the simulator was the following: environments and simulations are distributed through thread blocks, and rule blocks among threads. Phases 1, 3 and 4 were efficiently executed on the GPU, while Phase 2 was poorly accelerated given that it is inherently sequential. Concerning phase 3, it requires the creation of random binomial variate generation, what required the development of a new CUDA library [30] for binomial and multinominal random number generation, called cuRNG_BINOMIAL. It uses the normal approximation for large parameters values, and the BINV algorithm for low ones. For both situations, the cuRAND library was used.

The simulator is available in the PMCGPU project as a standalone tool, receives as input a file in binary format, and outputs a csv file with the information

collected for every transition and simulation performed. It was first benchmarked with a set of randomly generated PDP systems (without biological meaning), achieving speedups of up to 7x for large sizes on an NVIDIA Tesla C1060 GPU over the multi-core version. In [22], the authors validated the simulator by using a known ecosystem model of the Bearded Vulture in the Catalan Pyrenees [3]. The achieved speedup with this real ecosystem model, using the same very C1060 GPU, was up to 4.9x, and 18.1x using a K40 GPU (2880 cores).

3.5 Spiking Neural P systems

Cabarle et al. [4] initiated the first approaches to the parallel simulation of Spiking Neural P (SNP) systems. The key component of this is the simulation algorithm, which is based on a matrix representation of SNP systems. Introduced by Zeng et al. [29], the simulation of a SNP system can be carried out by considering the following vectors and matrices, aiming at providing a good representation for GPUs since they have been demonstrated to be well-suited for linear algebra:

- Spiking transition matrix: contains information about rules and how transitions are made. Assigns a row per rule and a column per neuron.
- Spiking vector: defines a selection of rules to be fired in a transition step, using a position per rule. Note that with non-determinism, it would be possible to have more than one spiking vector.
- Configuration vector: defines the number of spikes per neuron, that is, the configuration in a given time.

The simulator, codenamed CuSNP, was written in Python, and the CUDA kernels were launched by using the binding library PyCUDA. In the first approach, SNP systems without delays were simulated by covering each computation path in parallel, leading to speedups of up to 2.31x [5].

Recently, the authors developed a set of extensions to CuSNP [6], where, using a GTX 750 GPU, the achieved speedups are of up to 50x for very large instances:

- SNP systems with delays are supported. For this reason, the simulation algorithm, and hence the matrix representation, has been extended accordingly. In this case, a vector for neuron statuses and another for delays are introduced.
- More types of regular expressions are supported for the left-hand side of rules.
- P-Lingua files are supported as input, through their conversion into files in binary format [17].
- Validation of the simulators with models from the literature, such as a bitonic sorting network.

Lagunda et al. [17] implemented the same simulation algorithm for SNP systems using OpenCL. As mentioned, OpenCL is conceived to be a standard for GPU computing, being able to be executed in different kind of platforms, not only in NVIDIA GPUs. In their experiments, they only used a mobile GeForce 720M GPU, achieving up to 4.16x of speedup.

Macías-Ramos et al. [18] introduced a novel GPU simulator for Fuzzy Reasoning Spiking Neural (FRSN) P systems, which is a variant of SNP systems incorporating fuzzy logic elements. It allows modeling fuzzy diagnosis knowledge and reasoning for fault diagnosis applications. The simulation algorithm, as with SNP systems, is based on a matrix representation and vector-matrix operations. First, the simulator was implemented within the pLinguaCore simulation framework, written in Java. After validating the simulation algorithm, the core modules were deployed to the GPU and connected directly with pLinguaCore by using the binding library JCUDA.

3.6 Enzymatic Numerical P systems

García-Quismondo et al. [13] developed simulators for modeling robot controllers, being significant for the Artificial Intelligence. The implemented simulation algorithm reproduces the stages of an ENP system model: (1) Selection of applicable programs, (2) calculation of production functions and (3) distribution of production function results according to repartition protocols. Production functions are computed using a recursive solution. In general, the GPU design parallelizes the execution of programs among threads.

The simulators were implemented in Java (inside pLinguaCore) and C programming languages as standalone tools, being included in PMCGPU. On a GeForce GTX 460M, the achieved speedup was of up to 11x.

3.7 Evolution-Communication P systems with Energy

Initiated by Juayong et al. [14], the first simulator of Evolution-Communication P systems with Energy (ECPE) and without Antiport Rules made use of a matrix representation and linear-algebra based algorithm, similarly as for the spiking neural P systems simulator. In this case, a configuration vector (representing the frequency of objects in regions), a trigger matrix (objects satisfying a rule), an application vector (frequency of rule applications) and a transition vector (effect of rule applications) per region are used. The simulator was implemented in Python, and the GPU kernels execute the search for application vectors and model transition in parallel, by using PyCUDA.

In Barangan et al. [2], the work was extended in such a way that all application vectors were computed in parallel, leading to a solution that handles nondeterminism in an efficient way. In Argarin et al. [1], the simulator was also implemented in OpenCL and extended to support antiport rules, leading to speedups of up to 12x. As in the previous work, the GPU executes the corresponding kernels by using the binding library for Python and OpenCL, PyOpenCL

4 Conclusions and future work

Since the introduction of CUDA 10 years ago, many scientific applications have been accelerated. P system simulation is one of them: GPUs are good alternative

to conventional computing platforms due to the double parallel nature that both GPUs and P systems present. Their shared memory system also helps to efficiently synchronize the simulation of the models. Many authors have implemented simulators for their models of study. In order to do that, first, a simulation algorithm has to be defined, allowing a parallel implementation on GPUs.

However, it has been shown that P systems simulations are memory and memory-bandwidth bound. Indeed, simulating a P systems requires more data accesses than computing, and also a high synchronization degree (e.g., the global clock of the models). This fact restricts the design of parallel simulators, needing a careful representation and management of each P system ingredient. A bad step taken on GPU programming can easily break parallelism, and so, performance.

There is still a plethora of open lines concerning parallel simulation of P systems on GPUs. So far, the majority of simulators have been a proof of concept, where a good parallel design of the simulators was explored. However, none of them have been used in real applications yet, since this requires efficient communication protocols with more general, high-level simulation frameworks (such as P-Lingua). Moreover, further improvements to the designs can be done, taking advantage of the new features offered by the newest generation GPUs (such as the upcoming *thread groups* in CUDA 9 [30]). Finally, other P system models can be considered to be simulated on GPUs, as long as a simulation algorithm is well defined, accurate enough and parallelizable.

References

1. P.J. Argarin, N.J. Joaquin, R.A. Juayong, N.H. Hernandez, H.N. Adorna, F.G.C. Cabarle. An Implementation of Computations in Evolution-Communication P systems with Energy Using Open Computing Language. In *Proceedings of 16th Philippine Computing Science Congress*, 2016, pp. 63–76.
2. Z.F. Bangalan, K.A.N. Soriano, R.A.B. Juayong, F.G.C. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor. A GPU Simulation for Evolution-Communication P Systems with Energy Having no Antiport Rules, In *Proceedings of 11th Brainstorming Week on Membrane Computing*, 2013, pp. 25–50.
3. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A P system based model of an ecosystem of some scavenger birds, *Lecture Notes in Computer Science*, **5957**, (2010), 182–195.
4. F. G. Cabarle, H. N. Adorna, M. A. Martínez-del-Amor. A spiking neural P system simulator based on CUDA. *Lecture Notes in Computer Science*, 7184 (2012), 87–103.
5. F. G. Cabarle, H. N. Adorna, M. A. Martínez-del-Amor, M. J. Pérez-Jiménez. Improving GPU simulations of spiking neural P systems, *Romanian Journal of Information Science and Technology*, **15**, 1 (2012), 5–20.
6. J.P. Carandang, J.M.B. Villaflores, F.G.C. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor. CuSNP: Spiking Neural P Systems Simulators in CUDA. Romanian Journal of Information Science and Technology, **20**, 1 (2017), 57–70.
7. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with Active Membranes on CUDA, *Briefings in Bioinformatics*, **11**, 3 (2010), 313–322.

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*, **79**, 6 (2010), 317–325.
9. J. M. Cecilia, J. M. García, G. D. Guerrero, M. A. Martínez-del-Amor, M. J. Pérez-Jiménez, M. Ujaldón. The GPU on the simulation of cellular computing models, *Soft Computing*, **16**, 2 (2012), 231–246.
10. J. M. Cecilia, J. M. García, G. D. Guerrero, M. Ujaldón. Evaluating the SAT problem on P systems for different high-performance architectures, *The Journal of Supercomputing*, **69**, 1 (2014), 248–272.
11. M.A. Colomer-Cugat, M. García-Quismondo, L.F. Macías-Ramos, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, L. Valencia-Cabrera. Membrane system-based models for specifying Dynamical Population systems. In P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez (eds.), *Applications of Membrane Computing in Systems and Synthetic Biology. Emergence, Complexity and Computation series*, Volume **7**. Chapter 4, pp. 97–132, 2014, Springer Int. Publishing.
12. P. Frisco, M. Gheorghe, M. J. Pérez-Jiménez. *Applications of Membrane Computing in Systems and Synthetic Biology*, Series: Emergence, Complexity and Computation, **7**. Springer, 2014.
13. M. García-Quismondo, L. F. Macías-Ramos, M. J. Pérez-Jiménez. Implementing enzymatic numerical P systems for AI applications by means of graphic processing units. *Beyond Artificial Intelligence*, volume **4** of *Topics in Intelligent Engineering and Informatics*, 2013, pp. 137–159.
14. R. A. Juayong, F. G. Cabarle, H. N. Adorna, M. A. Martínez-del-Amor. On the simulations of evolution-communication P systems with energy without antiport rules for GPUs. *Tenth Brainstorming Week on Membrane Computing*, volume I, 2012, pp. 267–290.
15. M. Harris. Mapping computational concepts to GPUs, *ACM SIGGRAPH 2005 Courses*, NY (USA), 2005.
16. D. Kirk, W. Hwu. *Programming Massively Parallel Processors: A Hands On Approach*, MA (USA), 2010.
17. A.R. Lagunda, G.I. Palaganas, F.G.C. Cabarle, H Adorna. Spiking Neural P Systems GPU Simulation using OpenCL. In *Proceedings of 16th Philippine Computing Science Congress*, 2016, pp. 215–221.
18. L.F. Macías-Ramos, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez. Simulating FRSN P systems with real numbers in P-Lingua on sequential and CUDA platforms. *Lecture Notes in Computer Science*, **9504** (2015), 262–276.
19. M.A. Martínez-del-Amor. *Accelerating Membrane Systems Simulators using High Performance Computing with GPU*. Ph.D. thesis, University of Seville, 2013.
20. M.A. Martínez-del-Amor, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez. Simulating P Systems on GPU Devices: A Survey. *Fundamenta Informaticae*, **136**, 3 (2015), 269–284
21. M. A. Martínez-del-Amor, I. Karlin, R. E. Jensen, M. J. Pérez-Jiménez, A. C. Elster. Parallel simulation of probabilistic P systems on multicore platforms. In *Proceedings of the Tenth Brainstorming Week on Membrane Computing*, volume II, 2012, pp. 17–26.
22. M.A. Martínez-del-Amor, L.F. Macías-Ramos, L. Valencia-Cabrera, M.J. Pérez-Jiménez. Parallel simulation of Population Dynamics P systems: updates and roadmap. *Natural Computing*, **15**, 4 (2016), 565–573.

23. M.A. Martínez-del-Amor, J. Pérez-Carrasco, M.J. Pérez-Jiménez. Characterizing the parallel simulation of P systems on the GPU. *International Journal of Unconventional Computing*, **9**, 5-6 (2013), 405-424.
24. M.A. Martínez-del-Amor, I. Pérez-Hurtado, A. Gastalver-Rubio, A.C. Elster, M.J. Pérez-Jiménez. Population Dynamics P systems on CUDA. In *10th Conference on Computational Methods in Systems Biology, CMSB2012* (D. Gilbert, M. Heiner, eds.), LNBI 7605 (2012), 247-266.
25. M. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, Á. Romero-Jiménez, C. Graciani-Díaz, A. Riscos-Núñez, M.A. Colomer, and M.J. Pérez-Jiménez. DCBA: Simulating population dynamics P systems with proportional object distribution. *Lecture Notes in Computer Science*, **7762** (2013), 257-276.
26. A. Maroosi, R.C. Muniyandi, E.A. Sundararajan, A.M. Zin. Improved Implementation of Simulation for Membrane Computing on the Graphic Processing Unit, *Procedia Technology*, **11**, 2013, pp. 184-190.
27. Gh. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998
28. G. Păun, G. Rozenberg, A. Salomaa, eds. *The Oxford Handbook of Membrane Computing*, Oxford University Press, USA, 2010.
29. X. Zeng, H. Adorna, M. A. Martínez-del-Amor, L. Pan, M. J. Pérez-Jiménez. Matrix representation of spiking neural p systems. *Lecture Notes in Computer Science*, 6501 (2011), 377–391.
30. *NVIDIA CUDA website*, last accessed 2017. `https://developer.nvidia.com/cuda-zone`
31. *NVIDIA CUDA programming guide*, last accessed 2017. `http://docs.nvidia.com/cuda/cuda-c-programming-guide`
32. *Khronos Group OpenCL website*, last accessed 2017. `https://www.khronos.org/opencl`
33. *The PMCGPU project*, 2013. `http://sourceforge.net/p/pmcgpu`

Bibliographies

A Bibliography of Technological Applications of Spiking Neural P Systems

Gexiang Zhang[1,2], Qiang Yang[1], Linqiang Pan[3]

[1] Robotics Research Center, Key Laboratory of Fluid and Power Machinery of Ministry of Education, Xihua University, Chengdu, 610039, China
[2] School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031, China
[3] School of Automation, Huazhong University of Science and Technology, Wuhan 430074, China

This is a bibliography of technological applications of spiking neural P systems (SN P systems, for short) reported in the past years.

The study of SN P systems and of their applications is a very active and significant research topic. The bibliography may be useful and beneficial to the researchers in the community of membrane computing and related areas, especially to the researchers and students who are working on SN P systems.

Some of references in the list come from "A bibliography of spiking neural P systems" (L. Pan, T. Wu, Z. Zhang, *Bulletin of IMCS*, 2016, vol. 1, 63–78, online: `http://membranecomputing.net/ IMCSBulletin/`). There are also some updates. This list is meant to highlight the applications of SN P systems.

References

1. S. Aoki, A. Fujiwara: Asynchronous SN P systems for sorting. *Networking and Computing (ICNC), 2012 Third International Conference on. IEEE*, 2012, 221–225.
2. A. Binder, R. Freund, M. Oswald: Extended spiking neural P systems with astrocytes - variants for modelling the brain. *Proc. 13th Intern. Symp. AL and Robotics*, AROB2008, Beppu, Japan, 520–524.
3. R. Ceterchi, A.I. Tomescu: Spiking neural P systems, a natural model for sorting networks. *BWMC2008*, 93–106.
4. R. Ceterchi, A.I. Tomescu: Computing the maximum bisimulation with spiking neural P systems. *Computation, cooperation, and life*, Springer Berlin Heidelberg, 2011, 151–157.

5. R. Ceterchi, A.I. Tomescu: Implementing sorting networks with spiking neural P systems. *Fundamenta Informaticae*, 87, 1 (2008), 35–48.
6. H. Chen, X. Gao: Decimal Transforming Operations in Spiking Neural P Systems. *2nd International Conference on Biomedical Engineering and Informatics*, 2009.
7. K. Chen, J. Wang, Z. Sun, J. Luo, T. Liu: Programmable Logic Controller Stage Programming Using Spiking Neural P Systems. *Journal of Computational and Theoretical Nanoscience*, 12, 7 (2015), 1292–1299.
8. D. Díaz-Pernil, F. Peña-Cantillana, M. A. Gutiérrez-Naranjo: A parallel algorithm for skeletonizing images by using spiking neural P systems. *Neurocomputing*, 115 (2013), 81–91.
9. D. Díaz-Pernil, F. Peña-Cantillana, M. A. Gutiérrez-Naranjo: Skeletonizing images by using spiking neural P systems. *BWMC2010*, 91–103.
10. S. Elias, A. Chandar, K. G.Krithivasan, S. V. Raghavan: An Adaptive e- Learning Environment using Distributed Spiking Neural P Systems. *Technology for Education (T4E), 2011 IEEE International Conference on. IEEE*, 2011, 56–60.
11. R. Freund, M. Oswald: Regular ω-languages defined by extended spiking neural P systems. *Fundamenta Informaticae*, 83, 1–2 (2008), 65–73.
12. X. Gao, H. Chen: Signed integer arithmetic on spiking neural P systems. *Applied Mechanics and Materials*, 20 (2010), 779–784.
13. M.A. Gutiérrez-Naranjo, A. Leporati: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. *BWMC2008*, 193–210.
14. M.A. Gutiérrez-Naranjo, A. Leporati: Performing arithmetic operations with spiking neural P systems. *BWMC2009*, vol. I, 181–198.
15. M. A. Gutiérrez-Naranjo, A. Leporati: First steps towards a CPU made of spiking neural P systems. *Int. J. of Computers, Communications and Control*, 4, 3 (2009), 244–252.
16. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: A first model for Hebbian learning with spiking neural P systems. *BWMC2008*, 211–234.
17. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Hebbian learning from spiking neural P systems view. Proc. WMC9, Edinburgh, UK, 2008, 217–230.
18. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: A Spiking Neural P system based model for Hebbian Learning. *Proc. WMC9*, Edinburgh, UK, 2008, 189–207.
19. R. Hamabe, A. Fujiwara: Asynchronous SN P systems for logical and arithmetic operations. *Proceedings of International Conference on Foundations of Computer Science*, 2012, 58–64.
20. O.H. Ibarra, A. Păun, A. Rodriguez-Patón: Sequentiality induced by spike numbers in SN P systems. *Proc. 14th Intern. Meeting on DNA Computing*, Prague, June 2008, 36–46.
21. O.H. Ibarra, S. Woodworth: Characterizing regular languages by spiking neural P systems. *Intern. J. Found. Computer Sci.*, 18, 6 (2007), 1247–1256.
22. R. Idowu, R. Chandren, Z. Othman: Advocating the use of fuzzy reasoning spiking neural P systems in intrusion detection. *ACMC 2014*, 1–5.
23. R. Idowu, R. Muniyandi, Z. Othman: The Prospects of Using Spiking Neural P Systems for Intrusion Detection. *International Journal of Information and Network Security*, 2, 6 (2013), 492.
24. M. Ionescu, D. Sburlan: Some applications of spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 383–394, and *Computing and Informatics*, 27 (2008), 515–528.

25. M. Ionescu, C.I. Tîrnăucă, C. Tîrnăucă: Dreams and spiking neural P systems. *Romanian J. Inform. Sci. and Technology*, 12, 2 (2009), 209–217.
26. M. Ionescu, D. Sburlan: Some applications of spiking neural P systems. *Computing and Informatics*, 27, 3 (2012), 515–528.
27. T.-O. Ishdorj, A. Leporati: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing*, 7, 4 (2008), 519–534.
28. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with precomputed resources. *Theoretical Computer Sci.*, 411, 25 (2010), 2345–2358.
29. T.-O. Ishdorj, A. Leporati, L. Pan, J. Wang: Solving NP-Complete problems by spiking neural p systems with budding rules. *Proc. WMC10*, Curtea de Argeş, Romania, August 2009, 335–353.
30. Y. Kong, D. Zhao: Parallel Programming in Spiking Neural P Systems with Synapses States. *Journal of Computational and Theoretical Nanoscience*, 12, 10 (2015), 3418–3423.
31. A. Leporati, M. A. Gutiérrez-Naranjo: Solving Subset Sum by spiking neural P systems with pre-computed resources. *Fundamenta Informaticae*, 87, 1 (2008), 61–77.
32. A. Leporati, G. Mauri: Towards a High-Level Programming of Spiking Neural P Systems. *Emerging Paradigms in Informatics, Systems and Communication*, (2009), 99.
33. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform solutions to SAT and Subset-Sum by spiking neural P systems. *Natural computing*, 8, 4 (2009), 681–702.
34. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical NP-complete problems with spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 405–424.
35. X. Li, Z. Wang, W. Lu, Z. Chen, Y. Wang, X. Shi: A Spiking Neural System Based on DNA Strand Displacement. *Journal of Computational and Theoretical Nanoscience*, 12, 2 (2015), 298–304.
36. X. Liu, Z. Li, J. Liu, X. Zeng: Implementation of Arithmetic Operations With Time-Free Spiking Neural P Systems. *IEEE Transactions on NanoBioscience*, 14, 6 (2015), 617–624.
37. X. Liu, Z. Li, J. Suo, J. Liu, X. Min: A uniform solution to integer factorization using time-free spiking neural P system. *Neural Computing and Applications*, 26, 5 (2015), 1241–1247.
38. L. F. Macías-Ramos, M. A. Martínez-del-Amor, M.J. Pérez-Jiménez: Simulating FRSN P Systems with Real Numbers in P-Lingua on sequential and CUDA platforms. *16th International Conference on Membrane Computing 2015* (G. Rozenberg et al., eds.), LNCS 9504, 2015, Springer Berlin Heidelberg, 262–276.
39. V. P. Metta, A. Kelemenová: Sorting Using Spiking Neural P Systems with Anti-spikes and Rules on Synapses. *16th International Conference on Membrane Computing 2015* (G. Rozenberg et al., eds.), LNCS 9504, 2015, Springer Berlin Heidelberg, 290–303.
40. V.P. Metta, K. Krithivasan: Spiking neural P systems and Petri nets. *Proceedings of the International Workshop on Machine Intelligence Research*, 2009.
41. V. P. Metta, K. Krithivasan, D. Garg: Protocol Modeling in Spiking Neural P systems and Petri nets. *International Journal of Computer Applications*, 1, 24 (2010), 56–61.
42. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. *BWMC2009*, vol. II, 59–72.

43. T. Y. Nishida: Computing k-block Morphisms by Spiking Neural P Systems. *Fundamenta Informaticae*, 111, 4 (2011), 453–464.
44. A. Obtulowicz: Spiking neural P systems and modularization of complex networks from cortical neural network to social networks. *BWMC2009*, 109–114.
45. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems. Recent results, research topics. *Algorithmic Bioprocesses* (A. Condon, D. Harel, J. N. Kok, A. Salomaa, E. Winfree, eds.), Springer Berlin Heidelberg, 2009, 273–291.
46. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems. An overview. *Advancing Artificial Intelligence through Biological Process Applications* (A.B. Porto, A. Pazos, W. Buno, eds.), Medical Information Science Reference, Hershey, New York, 2008, 60–73.
47. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Computing morphisms by spiking neural P systems. *Intern. J. Found. Computer Sci.*, 18, 6 (2007), 1371–1382.
48. X. Peng, X. Fan, J. Liu: Performing Balanced Ternary Logic and Arithmetic Operations with Spiking Neural P Systems with Anti-Spikes. *Advanced Materials Research. Trans Tech Publications*, 505 (2012), 378–385.
49. H. Peng, J. Wang: Adaptive spiking neural P systems. *2010 Sixth International Conference of Natural Computing* (ICNC2010), 2010, vol. 6, 3008–3011.
50. H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang: Fuzzy reasoning spiking neural P system for fault diagnosis. *Information Sciences*, 235 (2013), 106–116.
51. M.J. Pérez-Jiménez: Simulating FRSN P Systems with Real Numbers in P-Lingua on sequential and CUDA platforms. *16th International Conference on Membrane Computing 2015* (G. Rozenberg et al., eds.), LNCS 9504, 2015, Springer Berlin Heidelberg, 262.
52. C. Qiu, L. Xiang, X. Liu: Broadcast Routing Algorithms in Hypercube Based on SN P Systems. *Pervasive Computing and the Networked World*, Springer International Publishing, 2013, 487–496.
53. D. Reid, M. Barrett-Baxendale: Spatiotemporal Processing in a Spiking Neural P System. *Proc. DESE'09*, 2009, 394–399.
54. T. Song, L. Luo, J. He, Z. Chen, K. Zhang: Solving subset sum problems by time-free spiking neural P systems. *Applied Mathematics & Information Sciences*, 8, 1 (2014), 327.
55. T. Wang, J. Wang, H. Peng, H. Wang: Knowledge representation and reasoning based on FRSN P system. *Intelligent Control and Automation (WCICA), 2011 9th World Congress on. IEEE*, 2011, 849–854.
56. T. Wang, T. Wang, H. Peng, Y. Deng : Knowledge representation using fuzzy spiking neural P system. *Proceedings of the IEEE Sixth International Conference on Bio-Inspired Computing: Theories and Applications*, Changsha, China, 2010, 586–590.
57. T. Wang, G. Zhang, M.J. Pérez-Jiménez: Application of weighted fuzzy reasoning spiking neural P systems to fault diagnosis in traction power supply systems of high-speed railways. *BWMC2014*, 329–350.
58. T. Wang, G. Zhang, M.J. Pérez-Jiménez: Fault diagnosis models for electric locomotive systems based on fuzzy reasoning spiking neural P systems. *15th International Conference on Membrane Computing 2014* (M. Gheorghe et al., eds.), LNCS 8961, Springer, 2014, 385–395.
59. T. Wang, G. Zhang, M.J. Pérez-Jiménez, J. Chen: Weighted fuzzy reasoning spiking neural P systems: application to fault diagnosis in traction power supply systems of

high-speed railways. *Journal of Computational and Theoretical Nanoscience*, 12, 7 (2015), 1103–1114.
60. T. Wang, G. Zhang, H. Rong, M.J. Pérez-Jiménez: Application of fuzzy reasoning spiking neural P systems to fault diagnosis. *International Journal of Computers Communications & Control*, 9, 6 (2014), 786–799.
61. T. Wang, G. Zhang, J. Zhao, J. Wang, M.J. Pérez-Jiménez: Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transactions on Power Systems*, 30, 3 (2015), 1182–1194.
62. J. Wang, H. J. Hoogeboom, L. Pan, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with weights. *Neural Computation*, 22, 10 (2010), 2615–2646.
63. J. Wang, H. J. Hoogeboom, L. Pan: Spiking neural P systems with neuron division. *11th International Conference on Membrane Computing 2010* (M. Gheorghe et al., eds.), LNCS 6501, Springer, 2010, 361–376.
64. J. Wang, H. J. Hoogeboom, L. Pan, Gh. Păun: Spiking neural P systems with weights and thresholds. *Proc. 10th Workshop Membrane Comput.*, Aug. 2009, 514–533.
65. J. Wang, H. Peng: Fuzzy knowledge representation based on an improving spiking neural P system. *2010 Sixth International Conference on Natural Computation* (ICNC2010), 2010, 3012–3015.
66. J. Wang, H. Peng: Adaptive fuzzy spiking neural P systems for fuzzy inference and learning. *International Journal of Computer Mathematics*, 90, 4 (2013), 857–868.
67. J. Wang, P. Shi, H. Peng, M.J. Pérez-Jiménez, T. Wang: Weighted fuzzy spiking neural P systems. *IEEE Transactions on Fuzzy Systems*, 21, 2 (2013), 209–220.
68. J. Wang, L. Zhou, H. Peng, G. Zhang: An extended spiking neural P system for fuzzy knowledge representation. *International Journal of Innovative Computing, Information and Control*, 7, 7 (2011), 3709–3724.
69. G. Xiong, D. Shi, L. Zhu, X. Duan: A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems. *Mathematical Problems in Engineering*, 2013 (2013).
70. L. Xu, P. Jeavons: Simple neural-like P systems for maximal independent set selection. *Neural computation*, 25, 6 (2013), 1642–1659.
71. J. Xue, X. Liu: Solving directed hamilton path problem in parallel by improved SN p system. *Pervasive Computing and the Networked World*, Springer Berlin Heidelberg, 2012, 689–696.
72. X. Zeng, C. Lu, L. Pan: A weakly universal spiking neural P system. *Mathematical and Computer Modelling*, 52, 11 (2010), 1940–1946.
73. X. Zeng, T. Song, L. Pan, X. Zhang: Spiking neural P systems for arithmetic operations. *Proc. IEEE Sixth International Conference on Bio-Inspired Computing: Theories and Applications*, Penang, Malaysia, 2011, 296–301.
74. X. Zeng, T. Song, X. Zhang, L. Pan: Performing four basic arithmetic operations with spiking neural P systems. *IEEE Transactions on NanoBioscience*, 11, 4, (2012), 366–374.
75. X. Zhang, T.-O. Ishdorj, X. Zeng, L. Pan: Solving PSPACE-complete problems by spiking neural P systems with pre-computed resources. Submitted, 2008.
76. G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez: An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*, 24, 5 (2014), 1440006.
77. X. Zhang, X. Zeng, B. Luo, J. Xu: Several applications of spiking neural P systems with weights. *Journal of Computational and Theoretical Nanoscience*, 9, 6 (2012), 769–777.

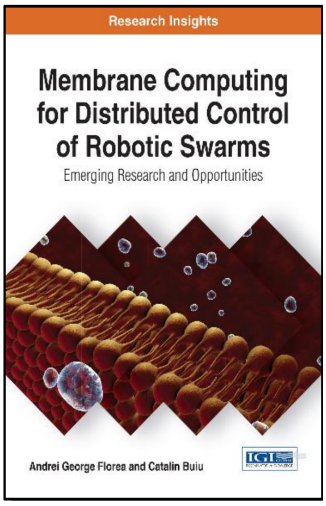
78. H. Peng, J. Wang, J. Ming, P. Shi, M. J. Pérez-Jiménez, W. Yu, C. Tao: Fault Diagnosis of Power Systems Using Intuitionistic Fuzzy Spiking Neural P Systems. *IEEE Transactions on Smart Grid*, accepted, doi:10.1109/TSG.2017.2670602.
79. J. Wang, H. Peng, M. Tu, M. J. Pérez-Jiménez, P. Shi: A Fault Diagnosis Method of Power Systems Based on an Improved Adaptive Fuzzy Spiking Neural P Systems and PSO Algorithms. *Chinese Journal of Electronics*, 25, 2(2016), 320–327.
80. T. Song, P. Zheng, M.L.D. Wong, X. Wang: Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control. *Information Sciences*, 372(2016), 380–391.
81. Z. Chen, P. Zhang, X. Wang, X. Shi, T. Wu, P. Zheng: A computational approach for nuclear export signals identification using spiking neural P systems. *Neural Computing and Applications*, 2016, 1–11.
82. C. Diaz, T. Frias, G. Sanchez, H. Perez, K. Toscano, G. Duchen: A novel parallel multiplier using spiking neural P systems with dendritic delays. *Neurocomputing*, 239(2017), 113–121.
83. K. Huang, T. Wang, Y. He, G. Zhang, M. J. Pérez-Jiménez: Temporal Fuzzy Reasoning Spiking Neural P Systems with Real Numbers for Power System Fault Diagnosis. *Journal of Computational and Theoretical Nanoscience*, 13, 6(2016), 3804–3814.
84. K. Huang, G. Zhang, X. Wei, H. Rong, Y. He, T. Wang: Fault Classification of Power Transmission Lines Using Fuzzy Reasoning Spiking Neural P Systems. *Bio-Inspired Computing-Theories and Applications*, 2016, 109–117.
85. Y. Yahya, A. Qian, A. Yahya: Power Transformer Fault Diagnosis Using Fuzzy Reasoning Spiking Neural P Systems. *Journal of Intelligent Learning Systems and Applications*, 8, 2016, 77–91.
86. J. Li, Y. Huang, J. Xu: Decoder Design Based on Spiking Neural P Systems. *IEEE Transactions on NanoBioscience*, 15, 7, 2016, 639–644.
87. G. Duchen, C. Diaz, G. Sanchez, H. Perez: First steps toward memory processor unit architecture based on SN P systems. *Electronics Letters*, 53, 6, 2017, 384–385.
88. C. Tao, W. Yu, J. Wang, P. Hong, K. Chen, J. Ming: Fault Diagnosis of Power Systems Based on Triangular Fuzzy Spiking Neural P Systems. *Bio-Inspired Computing-Theories and Applications*, 2016, 385–398.
89. J. Xue, X. Liu, P. Chen. Rhombic Grid Based Clustering Algorithm with Spiking Neural P Systems. *Journal of Computational and Theoretical Nanoscience*, 13, 6, 2016, 3895–3901.

Books Announcements

Membrane Computing for Distributed Control of Robotic Swarms: Emerging Research and Opportunities

Andrei George Florea, Cătălin Buiu

Laboratory of Natural Computing and Robotics
Department of Automatic Control and Systems Engineering
Politehnica University of Bucharest
Romania
{andrei.florea, catalin.buiu}@acse.pub.ro



We are pleased to announce the publication of our book *Membrane Computing for Distributed Control of Robotic Swarms: Emerging Research and Opportunities* by IGI Global (`http://www.igi-global.com/`), an U.S.A.-based publisher of Academic Research. The first author is Andrei Florea, a Ph.D. student and the main co-worker of prof. Cătălin Buiu (`http://catalin.buiu.net`), the second author. The foreword is written by Gheorghe Păun.

In 2011 we realized the first numerical P systems simulator, SNUPS. In 2012 we presented the first membrane controllers for mobile robots, that were based at that time on standard and enzymatic numerical P systems. In 2016 we introduced a new type of P colony, the XP colony, and the P swarm, a colony of XP colonies. Lulu, an open-source P colony/P swarm simulator (`https://github.com/andrei91ro/lulu_pcol_sim`) was also developed and presented in 2016.

After a brief introduction to swarm robotics, in Chapter 2 of the book we provide a theoretical overview of membrane computing that is completed with a

description of existing membrane computing simulators in Chapter 3. The core of the book is Chapter 4 in which we focus on the use of P colonies and P swarms for the control of single robots, but also for the control of multiple robots simultaneously. Execution diagrams are used to describe the behaviour of the membrane controllers and their effect on the state of the controlled robots. Some experiments that required large swarms (up to 15000 robots at 5x real speed, on a modern CPU) were executed using a port of Lulu in C that is small enough to be executed on the microcontroller of a real Kilobot robot. Each robot is controlled by one P colony or XP colony and all interactions are modelled using them. The last chapter provides a detailed guide on how to integrate membrane computing in mobile robotics applications in addition to other modelling and simulation tools and computational approaches.

There are many simulation examples presented in this new book, and part of the included experiments are validated on open-source, low cost mobile robots (Kilobot). The dedicated webpage (`http://membranecomputing.net/IGIBook/`) is an additional source of information where source code, input files, user manuals, and demonstration videos can be found.

The book, both in printed and electronic form, may be bought from `http://www.igi-global.com/book/membrane-computing-distributed-control-robotic/173023` (the coupon code **IGI40** will grant a 40% discount). The book is also available on Google Books (`https://books.google.ro/books?hl=en&lr=&id=E2xxDgAAQBAJ&oi=fnd&pg=PR1&ots=G19HceLr0C&sig=z0bXG3GJtKdk8kE8b-wR_RmATYA&redir_esc=y#v=onepage&q&f=false`).

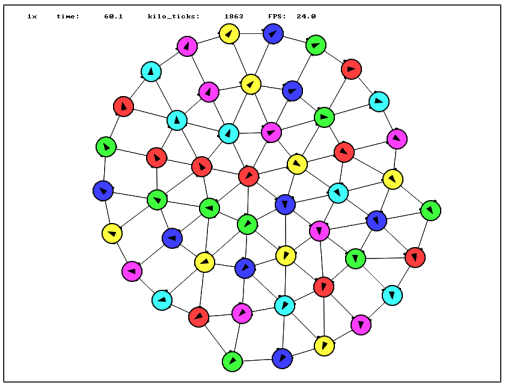


Fig. 1. An example simulation of a swarm of 50 Kilobot robots using the Kilombo simulator. Each robot is executing a time interrupt-driven P colony controller that allows them to chose a new direction every two seconds

We look forward to receiving your comments, suggestions and questions. We are also open for a future cooperation in this area.

Real-life Applications with Membrane Computing

Gexiang Zhang[1], Mario J. Pérez-Jiménez[2], Marian Gheorghe[3]

[1] Robotics Research Center,
Xihua University, Chengdu 610039, P.R. China; and
The Key Laboratory of Fluid and Power Machinery,
Xihua University, Ministry of Education, Chengdu 610039, P.R. China
gexiangzhang@gmail.com
[2] Department of Computer Science and Artificial Intelligence,
The University of Seville, Spain
marper@us.es
[3] School of Electrical Engineering and Computer Science,
The University of Bradford, UK
m.gheorghe@bradford.ac.uk

We are pleased to inform the Membrane Computing community about a newly published research monograph, with the title and authors mentioned above, in 2017, as Vol. 25 of the Emergence, Complexity and Computation series, with Springer. The book[4] has a preface and seven chapters, as mentioned below.

Table of contents

Chapter 0: Preface
Chapter 1: Membrane Computing - Key Concepts and Definitions
Chapter 2: Fundamentals of Evolutionary Computation
Chapter 3: Membrane Algorithms
Chapter 4: Engineering Optimization with Membrane Algorithms
Chapter 5: Electric Power System Fault Diagnosis with Membrane Systems
Chapter 6: Robot Control with Membrane Systems
Chapter 7: Data Modeling with Membrane Systems: Applications to Real Ecosystems

This book presents for the first time to the international community a complex set of real-life applications modeled and analyzed with a membrane computing apparatus consisting of methods and tools integrating previous membrane systems research threads as well as new developments. The most significant new research approaches include a combination of different P systems and evolutionary or fuzzy

[4] http://www.springer.com/series/10624

reasoning methods. The applications presented in the book cover a broad spectrum of topics, from various engineering areas, including engineering optimization, power systems fault diagnosis, mobile robots controller design, to complex biological systems involving data modeling and process interactions.

We will briefly present the topics investigated in each of the book chapters.

- In Chapter 1, *Membrane Computing - Key Concepts and Definitions*, are presented basic membrane computing concepts that are used in the models introduced in the next chapters. The most significant references to the research text books and overview papers are also provided.

- In Chapter 2, *Fundamentals of Evolutionary Computation*, are introduced fundamental concepts and principles of evolutionary computation. Five variants of evolutionary computation techniques, including genetic algorithms, quantum-inspired evolutionary algorithms, ant colony optimization, particle swarm optimization and differential evolution, are discussed. Details regarding their behavior, performances and usage are provided.

- In Chapter 3, *Membrane Algorithms*, are discussed a set of hybrid approximate optimization algorithms, called membrane algorithms or membrane-inspired evolutionary algorithms, integrating the hierarchical/network structure of P systems with meta-heuristic approaches. Most of these models have been introduced by the authors of the book. The design principles, their developments with key instances and examples are discussed. In addition, the impact of different variants of P systems with respect to membrane algorithms is analyzed.

- In Chapter 4, *Engineering Optimization with Membrane Algorithms,* a wide range of engineering applications of membrane algorithms with cell-like, tissue-like and neural-like P systems are discussed. The engineering problems include radar emitters signal analysis, digital image processing, controllers design, mobile robots path planning, constrained manufacturing parameters optimization problems, and distribution networks reconfiguration. Each model is presented with all the necessary details for implementing and analyzing its outcomes.

- In Chapter 5, *Electric Power System Fault Diagnosis with Membrane Systems,* Spiking Neural P systems incorporating fuzzy logics are utilized to solve fault diagnosis problems of electric power systems. Definitions, reasoning algorithms and examples of fuzzy reasoning spiking neural P systems are presented. The results are presented and comparisons with other approaches are mentioned.

- In Chapter 6, *Robot Control with Membrane Systems,* Numerical P (NP) systems and Enzymatic Numerical P (ENP) systems are employed for designing membrane controllers for mobile robots. Simulators for NP systems and ENP systems modeling the robots' behavior are described and the results analyzed.

– In Chapter 7, *Data Modeling with Membrane Systems: Applications to Real Ecosystems,* a bioinspired computing modeling paradigm within membrane computing, namely multienvironment P systems, is presented. This paradigm provides two different approaches (multicompartmental P systems and population dynamics P systems). The last approach is used to model population dynamics of real-world ecosystems. Ad-hoc algorithms and simulators are introduced to simulate, analyze and (experimentally) validate population dynamics P systems. These models target a wide class of applications, envisaging multi-disciplinary collaborations and aim at further extensions.

The book is of interest to a wider and diverse international audience, from researchers and academics working with natural and unconventional computational modeling, to PhD students looking for exciting, novel and challenging research topics, and from modelers and engineers interested in complex systems modeling and optimization, to researchers in biology, ecology, and more generally in natural sciences, interested in using mathematical and computational models for their own problems.

We hope that this book will provide to the researchers from the membrane computing community, especially to young and enthusiastic researchers and PhD students, a set of models and tools, as well as application areas, that will trigger further investigations into more complex and challenging applications of membrane computing and new interactions with other models and research areas.

Enjoy reading the book and do not hesitate to contact the authors when/if you have in mind interesting applications of membrane computing and are looking at some of the methods and tools presented by us. Any comments regarding this book, the methods introduced and the applications presented are welcome.

Open Problems

Some Open Problems on Numerical P Systems with Production Thresholds

Zhiqiang Zhang

Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

Summary. In this note, we propose some open problems on numerical P systems with production thresholds. First, the definition of numerical P systems with production thresholds is given, then some known results are shown, finally some problems are formulated.

1 Numerical P Systems with Production Thresholds

In this section, we give the definition of the computation model considered in this note. For more details, please refer to [1].

A numerical P system with production thresholds (in short, a TNP system) is a construct

$$\Pi = (m, L, \mu, T, (V_1, Pr_1, V_1(0)), \ldots, (V_m, Pr_m, V_m(0)), V_{in}, V_{out}),$$

where

- $m \geq 1$ is the number of membranes;
- L is an alphabet of labels for membranes in μ;
- μ is a rooted tree with m nodes labeled with the elements of L;
- T is a set of integers, the elements of which are called thresholds;
- $V_i = \{x_{j,i} \mid 1 \leq j \leq k_i\}$, $1 \leq i \leq m$, is the set of variables in region i;
- Pr_i, $1 \leq i \leq m$, is the set of programs in region i; each program has the following form:

  $$F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})|_{T_{l,i}} \rightarrow c_{l,i,1}|v_{l,i,1} + \ldots + c_{l,i,l_i}|v_{l,i,l_i},$$

  where $F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})$ is the production function, and $c_{l,i,1}|v_{l,i,1} + \ldots + c_{l,i,l_i}|v_{l,i,l_i}$ is the repartition protocol of the program; $T_{l,i} \in T$ is a constant;

- $V_i(0)$, $1 \leq i \leq m$, is the set of initial values of the variables in region i (the values of variables are integers);
- V_{in} and V_{out} are the sets of input and of output variables, respectively.

The programs allow the system to evolve the values of variables during computations. A *program* is composed of three parts: a *production function*, a *repartition protocol* and a *threshold*. Each program is evaluated in three phases, production-comparison-distribution.

- Production: The production function $F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})$ computes a value from the values of its variables at that time. $F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})$ can be any function using variables from the region that contains the program (only polynomial functions are considered here).
- Comparison: the production value is compared with the threshold $T_{l,i}$. Here, we consider two kinds of thresholds: bounding the production function value from below (lower-threshold) and bounding it from above (upper-threshold). More precisely, in the lower-threshold (resp., upper-threshold) case, if the production function value is greater (resp., smaller) than or equal to the threshold associated with it (the program is active), then the system proceeds to the distribution step; otherwise (the program is inactive), this production value is lost and no variable value is changed by this program at this step. For the active programs, after computing the production function value, the variables involved in the production function are reset to zero.
- Distribution: the production value is distributed to variables from the region where the program resides, and to variables in its upper (parent) and lower (children) compartments, as specified by the repartition protocol. Formally, for a repartition protocol $RP_{l,i}$, $\{v_{l,i,1}, \ldots, v_{l,i,l_i}\} \subseteq V_i \cup V_{par(i)} \cup (\bigcup_{ch \in Ch(i)} V_{ch})$, where $par(i)$ is the parent of membrane i and $Ch(i)$ is the set of children of membrane i. The coefficients $c_{l,i,1}, \ldots, c_{l,i,l_i}$ are natural numbers (they may also be 0, in this case the terms "$+0|x$" are omitted), which specify the proportion of the current production value distributed to each variable $v_{l,i,1}, \ldots, v_{l,i,l_i}$. At time t, if we denote with $C_{l,i} = \sum_{s=1}^{l_i} c_{l,i,s}$ the sum of all coefficients of the repartition protocol, and denote with

$$q_{l,i}(t) = \frac{F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))}{C_{l,i}} \tag{1}$$

the "unitary portion", then the value $ad_{l,i,r}(t) = q_{l,i}(t) \cdot c_{l,i,r}$ represents the value added to variable $v_{l,i,r}$. If variable $v_{l,i,r}$ appears in several repartition protocols $RP_{l_1,i_1}, \ldots, RP_{l_k,i_k}$ of the applied programs, then all these values $ad_{l_1,i_1,r}, \ldots, ad_{l_k,i_k,r}$ are added to variable $v_{l,i,r}$. So, if at time t variable $v_{l,i,r}$ is involved in at least one production function of the applied programs and appears in several repartition protocols $RP_{l_1,i_1}, \ldots, RP_{l_k,i_k}$ of the applied programs, then its value at time $t+1$ is $v_{l,i,r}(t+1) = \sum_{s=1}^{k} ad_{l_s,i_s,r}(t)$; otherwise, if at time t variable $v_{l,i,r}$ appears only in some repartition protocols

$RP_{l_1,i_1}, \ldots, RP_{l_k,i_k}$ of the applied programs not in any production function of the applied programs, then $v_{l,i,r}(t+1)=v_{l,i,r}(t) + \sum_{s=1}^{k} ad_{l_s,i_s,r}(t)$.

Note that the three phases, production-comparison-distribution, take place in one time unit.

TNP systems can evolve in one of the following modes:

- *sequential*: at each step, only one program is applied in each membrane; if more than one program in a membrane can be used, then one of them is non-deterministically chosen;
- *all-parallel*: at each step, in each membrane, all applicable programs are applied, allowing that more than one program share the same variable;
- *one-parallel*: apply programs in the all-parallel mode with the restriction that one variable can appear in only one of the applied programs; in the case of multiple choices, the programs to be applied are chosen in the non-deterministic way.

Initially, the variables have the values specified by $V_i(0), 1 \leq i \leq m$. At time $t \in N$, the values of all the variables of Π are defined as the *configuration* of Π at time t. Using programs in the way mentioned above, and choosing the program in the all-parallel or sequential mode, we obtain *transitions* among configurations. A sequence of such transitions forms a *computation*. If no applicable set of programs produces a change in the current configuration, we say that the system reaches a *final* or a *halting configuration*. (Because of thresholds, it is also possible to have halting computations in the standard sense, with no rule applicable.)

In this way, a numerical P system can compute a function f: $N^{\alpha} \rightarrow N^{\beta}$ $(\alpha, \beta \geq 0)$: the α values of the arguments are introduced in the system as the initial values of variables in V_{in} and the β-vector of the function value is obtained in the variables from V_{out} in the halting configuration of the system. If the system never reaches a halting configuration, then no result is obtained.

By ignoring the input variables (V_{in} is then omitted), (non-deterministic) numerical P systems with production thresholds can also be used in the *generating mode*, whereas by ignoring the output variables (V_{out} is then omitted) we can use (deterministic or non-deterministic) numerical P systems with production thresholds in the *accepting mode*.

Note that in equation (1), $q_{l,i}(t)$ is an integer only if the value of the production function $F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))$ is divisible by the respective sum $C_{l,i}$. If at any step, all the values of the production functions are divisible by the respective sum, we associate this kind of systems with the notation *div*. If a current production is not divisible by the associated coefficients total, then we can take the following decisions: (i) the remainder is lost (the production which is not immediately distributed is lost), (ii) the remainder is added to the production obtained in the next step (the non-distributed production is carried over to the next step), (iii) the system simply stops and aborts, no result is associated with that computation. We denote these three cases with *lost, carry, stop*, respectively. In this work, the

TNP systems are of the *div* type. In fact, for NP systems of the non-div type, without threshold, the computational university has been proved.

The set of natural numbers generated or accepted in the way mentioned above by a system Π is denoted by $N_\eta(\Pi), \eta \in \{gen, acc\}$. We use $N_\eta T^\gamma_{m_1} P^D_{m_2}(poly^n(r), \nu)$ to denote the family of all sets $N_\eta(\Pi)$ of numbers computed by systems Π working in η mode, with m_1 thresholds used in the $\gamma \in \{l, u\}$ way, with l indicating the lower-threshold and u indicating the upper-threshold case; with at most m_2 membranes, production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial, using the rules in the mode $\nu \in \{all, seq\}$, where *all* stands for all-parallel, and *seq* stands for sequential; the letter D indicates the use of deterministic systems (we remove D when the systems may also be non-deterministic). If one of the parameters m_1, m_2, n, r is not bounded, then it is replaced with $*$.

2 Known Results

The following results are obtained in [1] (NRE is the family of Turing computable sets of natural numbers).

Theorem 1. $N_{acc}T^l_4 P^D_1(poly^1(2), all) = N_{acc}T^u_4 P^D_1(poly^1(2), all) = NRE.$

Theorem 2. $N_{gen}T^l_5 P_1(poly^2(3), seq) = N_{gen}T^u_6 P_1(poly^2(3), seq) = NRE.$

Corollary 2.1 $N_{acc}T^l_5 P^D_1(poly^2(3), seq) = N_{acc}T^u_6 P^D_1(poly^2(3), seq) = NRE.$

3 Open problems

In this section, we list the following open problems for further investigation.

- It is proved that for NPT systems with lower-thresholds and upper-thresholds working both in the all-parallel mode and in the sequential mode, one membrane suffices to reach universality. The case of the one-parallel computation remains open.
- It remains open whether the number of thresholds used in the systems constructed in the proofs given in [1] can be decreased. In particular, it is of interest to investigate whether the system can reach universality by using the same threshold for all programs – we have a unique threshold for all variables.
- In the definition of numerical P systems with production thresholds, if the production function value is greater (resp., smaller) than or equal to the threshold associated with it in the case of lower-threshold (resp., upper-threshold), then the system proceeds to the distribution step. It deserves to check whether the computation power of systems changes if thresholds are used in the strict way (that is, a program can be applied only when the production value is strictly greater or smaller than the threshold). It is also of interest to investigate the

computation power of systems when thresholds are used in the mixed way (that is, in a system, some programs are associated with lower-thresholds and some programs are associated with upper-thresholds).

- Generally, numerical P systems working in the sequential mode need more than one membrane to reach computational universality. However, with the feature of threshold, all the universal numerical P systems working in the sequential mode constructed in [1] have only one membrane, which is similar to some sort of parallel rewriting grammar without actually exploiting the typical feature of membranes. It is of interest to consider whether we can construct universal numerical P systems with thresholds having less programs when several membranes are used.

References

1. Linqiang Pan, Zhiqiang Zhang, Tingfang Wu, Jinbang Xu, Numerical P Systems with Production Thresholds, to appear in *Theoretical Computer Science*.

Open Problems on Symport/Antiport (Tissue) P Systems with Channel States

Bosheng Song

Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China

Communication is an important feature of P systems that allows objects to move from one cell (or environment) to another region. Symport/antiport rules are typical examples of communication rules, which are inspired from the active transport of molecules across the membrane. P systems with symport/antiport rules were introduced in [5], where several results concerning computational completeness were obtained and then extended/improved in [1, 3].

The notion of channel states was first proposed in tissue P systems with symport/antiport rules [2], and several results about Turing universality and non-universality of symport/antiport tissue P systems are obtained considering as parameters the number of cells, the number of channel states and the length of symport/antiport rules [2, 7]. In [2], rules in a system are used in a sequential manner at the level of each channel (for each channel associated with two neighboring regions, at most one rule can be used at one step) and in a parallel manner at the level of the system (all channels which can use a rule must do it). In [4], flat maximal parallelism of using rules was proposed: in each step, in each membrane, a maximal set of applicable rules is chosen and each rule in the set is applied exactly once. Tissue P systems with channel states and symport/antiport rules working in the flat maximally parallel way were considered in [9], and the computation power of such P systems was investigated considering as parameters the number of cells, the number of channel states and the length of symport/antiport rules.

In [8], channel states were introduced into cell-like P systems with symport/antiport rules, where at most one channel is established between neighboring regions, each channel is associated with one state in order to control communication at each step, and rules are used in the same manner as in [2] (a sequential manner at the level of each channel and in a parallel manner at the level of the system).

In what follows, we formulate some open problems about symport/antiport (tissue) P systems with channel states for further research.

- The rules of cell-like P systems with channel states and symport/antiport rules are used in a sequential way on each channel. What is the computation power of cell-like P systems with channel states and symport/antiport rules that use rules on channels in other strategies, for instance, asynchronous, flat maximal parallelism, etc?
- In symport/antiport (tissue) P systems with channel states, there is at most one channel between two cells or between a cell and the environment. What is the computation power of symport/antiport (tissue) P systems with channel states if there are two or more channels between two cells or between a cell and the environment?
- The descriptional complexity of P systems with symport/antiport rules with respect to the number of objects was considered in [6]. It is of interest to investigate universal (tissue) P systems with channel states and symport/antiport rules with a small number of objects (following, for instance, the ideas of [6]).
- A usual idea for solving hard computational problems is to add the membrane division or membrane separation rules into a P system, an exponential space in a linear time can be produced and calculating in parallel for all possible candidate solutions. It would be interesting to introduce a variant of symport/antiport P systems with channel states that can have exponential channel states in linear time, thus solving **NP**-complete problems by using the exponentially many channel states instead of the exponentially many membranes generated by membrane division or membrane separation.

References

1. A. Alhazov, R. Freund, P systems with one membrane and symport/antiport rules of five symbols are computationally complete, in: Proc. of the Third Brainstorming Week on Membrane Computing, 2005, pp. 19–28.
2. R. Freund, Gh. Păun, M. J. Pérez-Jiménez, Tissue P systems with channel states, Theor. Comput. Sci. 330, 2005, 101–116.
3. P. Frisco, H. J. Hoogeboom, P Systems with symport/antiport simulating counter automata, Acta Inform. 41, 2004, 145–170.
4. L. Pan, Gh. Păun, B. Song, Flat maximal parallelism in P systems with promoters, Theor. Comput. Sci. 623, 2016, 83–91.
5. A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport. New Generat. Comput. 20(3), 2002, 295–305.
6. Gh. Păun, J. Pazos, M. J. Pérez-Jiménez, A. Rodríguez-Patón, Symport/antiport P systems with three objects are universal, Fund. Informa. 64, 2005, 1–4.
7. Gh. Păun, G. Rozenberg, A. Salomaa, (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, New York, 2010.
8. B. Song, L. Pan, M. J. Pérez-Jiménez, Cell-like P systems with channel states and symport/antiport rules, IEEE Trans. NanoBiosci. 15(6), (2016), 555–566.
9. B. Song, M. J. Pérez-Jiménez, Gh. Păun, L. Pan, Tissue P systems with channel states working in the flat maximally parallel way, IEEE Trans. NanoBiosci. 15(7), (2016), 645–656.

Calls for Participation to MC and Related Conferences/Meetings

18th International Conference on Membrane Computing (CMC18)

Bradford, United Kingdom

24-28 July, 2017

`http://computing.brad.ac.uk/cmc18/`

IMPORTANT DATES

- Deadline for submissions: 17 April 2017
- Notification of acceptance: 29 May 2017
- Final version: 19 June 2017
- Conference: 24 - 28 July 2017

SCOPE AND LOCATION

The Conference on Membrane Computing (CMC) series started in 2000 as the Workshop on Multiset Processing. The first Workshop on Membrane Computing was organized in Curtea de Argeş, Romania, in 2001. In 2010 it was transformed into a conference, CMC11. The last edition, CMC17, was held in Milan, Italy, in 2016. Nowadays a Steering Committee takes care of the continuation of the CMC series which is organized under the auspices of the International Membrane Computing Society (IMCS).

The 18th edition of the International Conference on Membrane Computing series will take place at the University of Bradford, UK, one of the top ten greenest universities in the world. Bradford is a multicultural, vibrant, and well-connected city in the heart of Yorkshire. Full of history, it was once the wool capital of the world, also the first UNESCO City of Film. Home to the National Media Museum,

Bradford is famous for some of the finest Asian food in the UK, being crowned Curry Capital of Britain for six consecutive years! It benefits from a maritime climate, with limited seasonal temperature ranges, and generally moderate rainfall throughout the year.

The goal of CMC18 is to bring together researchers working in membrane computing and related fields, in a friendly atmosphere enhancing communication, cooperation and continuing the tradition of the past meetings. Membrane computing (P systems theory) is an area of computer science aiming to abstract computing ideas and models from the structure and the functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures.

CONFERENCE FORMAT

This edition aims to have the following format, although some changes might occur based on suggestions made by the Steering Committee. It is planned to include: (1) three days of communications with invited speakers and short and long contribution talks according to the papers submitted and the reviews of the Program Committee, and (2) an Interaction Day similar in spirit to the Brainstorming Week on Membrane Computing (BWMC) that is usually organized in Sevilla every year, where the participants will work in a collaborative way to address open problems and propose new approaches, problems and results. An IMCS General Assembly will be also organized.

SUBMISSION OF PAPERS

Authors are invited to submit papers presenting original, unpublished research in PDF format. There are two tracks for submission:

1. full paper (of a reasonable length),
2. extended abstract for poster presentation (maximum four pages). Typical extended abstracts present significant work-in-progress, late-breaking results, or contributions from students new in the field or at the start of their research career.

Only electronic submissions are accepted. Papers should be written in LaTeX and formatted according to the usual LNCS article style which can be downloaded at Springer's LNCS website (`http://www.springer.com/lncs`). Please, include all source files as well as all additional files (figures etc.), and also attach a PDF version of the submission.

Submissions have to be sent through the EasyChair web page `https://easychair.org/conferences/?conf=cmc18`.

PROCEEDINGS

A pre-proceedings volume will be available at the conference in electronic format, online, and optionally hardcopies. A volume devoted to selected and additionally revised papers will be published in the Lecture Notes in Computer Science series of Springer-Verlag after the event.

STEERING COMMITTEE

Henry Adorna (Quezon City, Philippines)
Artiom Alhazov (Chişinău, Moldova)
Bogdan Aman (Iaşi, Romania)
Matteo Cavaliere (Edinburgh, Scotland)
Erzsébet Csuhaj-Varjú (Budapest, Hungary)
Giuditta Franco (Verona, Italy)
Rudolf Freund (Wien, Austria)
Marian Gheorghe (Bradford, UK) - Honorary Member
Thomas Hinze (Cottbus, Germany)
Florentin Ipate (Bucharest, Romania)
Shankara N. Krishna (Bombay, India)
Alberto Leporati (Milan, Italy)
Taishin Y. Nishida (Toyama, Japan)
Linqiang Pan (Wuhan, China)
Gheorghe Păun (Bucharest, Romania) - Honorary Member
Mario J. Pérez-Jiménez (Sevilla, Spain)
Agustín Riscos-Núñez (Sevilla, Spain)
José M. Sempere (Valencia, Spain)
Petr Sosík (Opava, Czech Republic)
Kumbakonam Govindarajan Subramanian (Penang, Malaysia)
György Vaszil (Debrecen, Hungary)
Sergey Verlan (Paris, France)
Claudio Zandron (Milan, Italy) - Chair
Gexiang Zhang (Chengdu, China)

PROGRAM COMMITTEE

Henry Adorna (Quezon City, Philippines)
Artiom Alhazov (Chişinău, Moldova)
Bogdan Aman (Iasi, Romania)
Lucie Ciencialová (Opava, Czech Republic)
Erzsébet Csuhaj-Varjú (Budapest, Hungary)
Giuditta Franco (Verona, Italy)

Rudolf Freund (Wien, Austria)
Thomas Hinze (Cottbus, Germany)
Marian Gheorghe (Bradford, UK)
Florentin Ipate (Bucharest, Romania)
Shankara N. Krishna (Bombay, India)
Alberto Leporati (Milan, Italy)
Vincenzo Manca (Verona, Italy)
Giancarlo Mauri (Milan, Italy)
Radu Nicolescu (Auckland, New Zealand)
Linqiang Pan (Wuhan, China)
Gheorghe Păun (Bucharest, Romania)
Mario J. Pérez-Jiménez (Sevilla, Spain)
Antonio E. Porreca (Milan, Italy)
Agustín Riscos-Núñez (Sevilla, Spain)
José M. Sempere (Valencia, Spain)
Petr Sosík (Opava, Czech Republic)
György Vaszil (Debrecen, Hungary)
Sergey Verlan (Paris, France)
Claudio Zandron (Milan, Italy)
Gexiang Zhang (Chengdu, China)

ORGANIZING COMMITTEE

Marian Gheorghe (Bradford, UK) - Co-chair
Savas Konur (Bradford, UK) - Co-chair
Raluca Lefticaru (Bradford, UK) - Communication Chair
Daniel Neagu (Bradford, UK)

CONTACT INFO

Please do not hesitate to contact us if you have any question.
Marian Gheorghe (email: `m.gheorghe@bradford.ac.uk`)
Savas Konur (email: `s.konur@bradford.ac.uk`)

The 6th Asian Conference on Membrane Computing (ACMC2017) 21-25 September, 2017 Chengdu (P.R. China)

Website: 2017.asiancmc.org

Call for papers

ACMC 2017 is one of the flagship conferences on Membrane Computing, aiming to provide a high-level international forum for researchers working in membrane computing and related areas, especially for the ones from the Asian region. This conference is the sixth edition of ACMC with the five editions having successfully taken place in Wuhan (China, 2012), Chengdu (China, 2013), Coimbatore (South India, 2014), Anhui (China, 2015), and Bangi (Malaysia, 2016), and also a geographical expansion of CMC (Conference on Membrane Computing) which is held every year from the year of 2000 in different European Countries.

Like artificial neural networks, evolutionary algorithms, swarm intelligence, cellular automata and DNA computing, membrane computing is also a branch of natural computing or nature-inspired computing and was initiated by Gheorghe Păun in 1998. It aims to abstract computing models, called membrane systems or P systems, from the structure and the functioning of the living cell as well as from the cooperation of cells in tissues, organs, and populations of cells. This research area has grown into a vigorous scientific discipline and has attracted a large number of researchers all over the world.

ACMC 2017 is planned as a friendly interactive conference with several introductory tutorials, several keynote lectures and some specialized sessions, which will cover a wide range of topics on membrane computing, including theory, applications, implementation and various aspects related to membrane computing. The registration fee will be of about 3500 RMB per person. This fee covers the participation, the special dinner, the tourist programme, as well as the lunches. The accommodation, to be paid by each participant, will be available at various prices. Various accommodation possibilities will be posted soon on the conference page.

Topics

The Sixth Asian Conference on Membrane Computing (ACMC2017) provides an open platform to bring together scholars worldwide to present their recent work on membrane computing. The topics of this conference are as follows (but not limited to):

(1) Theoretical aspects of membrane computing

- Various variants of computing models: cell-, tissue- and neural-like P systems.
- Computing power of membrane computing models.
- Computing efficiency of membrane computing models.

(2) Applications of membrane computing

- Robots controller design.
- Modeling using P systems for biological systems, ecological systems, etc.
- Membrane-inspired optimization algorithms for various problems.
- Fault diagnosis of various systems, such as robots, power systems, etc.
- Other applications.

(3) Implementation of membrane computing models

- Software implementation.
- Hardware implementation.
- Biological implementation.
- Other implementation.

Awards

ACMC 2017 sets up two best papers awards, BEST PAPER AWARD and BEST STUDENT PAPER AWARD, which aims to promote the academics, encourage young scientists to participate in academic activities, further to improve the paper quality and expand conference influence.

1. Best Paper Award Regulations:

(1) Eligibility

The considered papers must satisfy:

(i) The paper is accepted by the ACMC 2017;

(ii) One of the authors must register;

(iii) One of the authors must present the paper at the conference.

(2) Requirement

The paper should present a significant contribution with regard to theoretical results, applications or implementation of P systems, or comprehensive and high-level surveys on a specific topic, and should be written in a professional way.

2. Best Student Paper Award Regulations

(1) Eligibility

The considered papers must satisfy:

(i) The paper is accepted by the ACMC 2017;

(ii) The first author must be an undergraduate, master or PhD student.

(iii) The student must finish the registration process and present the paper at the conference.

(2) Requirement

The paper should present a significant contribution with regard to theoretical results, applications or implementation of P systems, or comprehensive and high-level surveys on a specific topic, and should be written in a professional way.

3. Procedure

The best paper and best student paper are selected by a Technical Committee through evaluating the reviewing reports, quality and presentation.

4. Award

The best papers elected by these regulations are awarded:

(1) A certificate of "ACMC 2017 Best Paper Award" or "ACMC 2017 Best Student Paper Award". The certificate is signed by the conference chair;

(2) Prize or gifts.

Invited Speakers

Mario J. Pérez-Jiménez, University of Sevilla, Spain
Marian Gheorghe, University of Bradford, UK
Ferrante Neri, De Montfort University, UK
Linqiang Pan, Huazhong University of Science and Technology, China
Sergey Verlan, University of Paris Est, France
Kumbakonam Govindarajan Subramanian, Madras Christian College, India
Jun Wang, Xihua University, China

Publications

Papers accepted for presentation will appear in conference pre-proceedings of ACMC 2017. After the conference, all the accepted papers will be selected to be considered for publication in the following reputed publications:

(1) A selection of the ACMC2017 accepted papers will be re-reviewed and published in the SCI-indexed journal *International Journal of Computers, Communications and Control* (IJCCC);

(2) A selection of the ACMC2017 accepted papers will be combined with the CMC2017 papers to be re-reviewed and published in *Lecture Notes in Computer Science* (LNCS) (EI-indexed volume), Springer;

(3) A selection of the ACMC2017 accepted papers will be combined with the BIC-TA2017 papers to be re-reviewed and published in *Communications in Computer and Information Science* (CCIS) (EI-indexed volume), Springer;

(4) A selection of the ACMC2017 accepted papers will be re-reviewed and published in the Springer journal *Journal of Membrane Computing* (JMC);

(5) More SCI-indexed journals are under discussion.

Important Dates

Submission deadline: **Sunday July 23, 2017**
Acceptance notification: Wednesday August 23, 2017
Camera-ready version and early registration: Sunday September 3, 2017

Submission

Authors are invited to submit their original research contributions (including significant work in progress) on membrane computing, its applications and related subjects. Papers (of reasonable length) should be formatted according to Lecture Notes in Computer Science (LNCS) format (please refer to **http://acmc2013.org/file/LatexTemplate.zip**). All papers should be submitted as PDF files through EasyChair conference system website. The submission Web page for ACMC 2017 is **https://easychair.org/conferences/?conf=acmc2017**.

If there is any difficulty or problem, please do not hesitate to contact the organizer by email: *gexiangzhang@gmail.com*.

Host institutions

Xihua University, Chengdu, China
Southwest Jiaotong University, Chengdu, China

Sponsors

International Membrane Computing Society (IMCS)

Xihua University, Chengdu, China

Committees

Steering Committee:
Henry Adorna (Quezon City, Philippines)
Artiom Alhazov (Chisinau, Moldova)

Bogdan Aman (Iasi, Romania)
Matteo Cavaliere (Edinburgh, Scotland)
Erzsébet Csuhaj-Varjú (Budapest, Hungary)
Giuditta Franco (Verona, Italy)
Rudolf Freund (Wien, Austria)
Marian Gheorghe (Bradford, UK) - Honorary member
Thomas Hinze (Cottbus, Germany)
Florentin Ipate (Bucharest, Romania)
Shankara N. Krishna (Bombay, India)
Alberto Leporati (Milan, Italy)
Taishin Y. Nishida (Toyama, Japan)
Linqiang Pan (Wuhan, China) – Co-Chair
Gheorghe Păun (Bucharest, Romania) - Honorary member
Mario J. Pérez-Jiménez (Sevilla, Spain)
Agustín Riscos-Núñez (Sevilla, Spain)
José M. Sempere (Valencia, Spain)
Petr Sosík (Opava, Czech Republic)
Kumbakonam Govindarajan Subramanian (Chennan, India)
György Vaszil (Debrecen, Hungary)
Sergey Verlan (Paris, France)
Claudio Zandron (Milan, Italy) – Co-Chair
Gexiang Zhang (Chengdu, China)

Program Committee:
Henry Adorna (Quezon City, Philippines)
Cătălin Buiu (Bucharest, Romania)
Matteo Cavaliere (Edinburgh, Scotland)
Erzsébet Csuhaj-Varjú (Budapest, Hungary)
Xiaoju Dong (Shanghai, China)
Marian Gheorghe (Bradford, UK)
Ping Guo (Chongqi, China)
Thomas Hinze (Cottbus, Germany)
Florentin Ipate (Bucharest, Romania)
Alberto Leporati (Milan, Italy)
Xiyu Liu (Jinan, China)
Taishin Nishida (Toyama, Japan)
Tseren-Onolt Ishdorj (Mongolia)
Jose M. Sempere (Valencia, Spain)
Ravie Chandren Muniyandi (Bangi, Malaysia)
Radu Nicolescu (Auckland, Australian)
Linqiang Pan (Wuhan, China) – Chair
Gheorghe Păun (Bucharest, Romania)
Hong Peng (Chengdu, China)
Mario J. Pérez-Jiménez (Sevilla, Spain)

Agustín Riscos-Núñez (Sevilla, Spain)
Tao Song (Qingdao, China)
Kumbakonam Govindarajan Subramanian (Chennan, India)
D.G. Thomas (Chennai, India)
Sergey Verlan (Paris, France)
Jun Wang (Chengdu, China)
Jianhua Xiao (Tianjing, China)
Hsu-Chun Yen (Taiwai, China)
Claudio Zandron (Milan, Italy)
Xiangxiang Zeng (Xiamen University)
Gexiang Zhang (Chengdu, China) – Co-Chair
Xingyi Zhang (Anhui, China)
Xue Zhang (Boston, USA)
Xuncai Zhang (Zhengzhou, China)

Organizing Committee:
Qingyou Liu (Xihua University, China) – General Chair
Gexiang Zhang (Xihua University, China, gexiangzhang@gmail.com) – Chair
Jun Wang (Xihua University, China) – Co-Chair
Wei Yang (Xihua University, China)
Zhongfan Xiang (Xihua University, China)
Xiantai Gou (Southwest Jiaotong University, China)
Haina Rong (Southwest Jiaotong University, China)
Zhi'ang Wan (Xihua University, China)
Qiang Yang (Xihua University, China)
Tao Ren (Xihua University, China)
Yujia Li (Xihua University, China)
Yu Wang (Xihua University, China)
Xiaoxiao Song (Xihua University, China)
Youxing Zeng (Xihua University, China)
Xucai Zeng (Xihua University, China)

Contact

Conference Secretaries
Qiang Yang, Youxing Zeng, Xucai Zeng (Xihua University, China)
No. 999, Jinzhou Road, Jinniu District, Chengdu, China
E-mail: qiangychd@126.com (English), zengyouxinxhu@163.com (Chinese), 1084948226@qq.com (Chinese)

15th International Conference on Automata and Formal Languages

Debrecen, Hungary

September 4-6, 2017

`http://www.inf.unideb.hu/afl2017`

Important dates:

Paper submission deadline: May 15, 2017.
Author notification: July 1, 2017.
Camera-ready deadline: July 15, 2017
Conference: September 4-6, 2017.

Scope, location, format

The AFL series, initiated by the late Professor István Peák in 1980, has a long tradition. The AFL conferences cover all aspects of automata and formal languages, including theory and applications. In 2017, the conference is organized by the Faculty of Informatics of the University of Debrecen and the Faculty of Informatics of the Eötvös Loránd University of Budapest, Hungary.

Topics of interest include (but are not limited to):

- Grammars, acceptors and transducers for strings, trees, graphs, arrays, etc.,
- algebraic theories for automata and languages,
- combinatorial properties of words and languages,
- formal power series,
- decision problems,
- efficient algorithms for automata and languages.
- Relations to
  - complexity theory and logic,
  - picture description and analysis,
  - quantum computing,
  - cryptography,

  - concurrency.
- Applications of automata and language theory in
  - biology,
  - natural language processing,
  - and other fields.

The scientific program will consist of invited lectures and contributed presentations selected by the international Program Committee.

Invited speakers:

- Paola Bonizzoni (Università di Milano-Bicocca, Milan, Italy)
- Henning Bordihn (University of Potsdam, Germany)
- Szilárd Zsolt Fazekas (Akita University, Japan)
- José M. Sempere (Technical University of Valencia, Spain)
- Akihiro Yamamura (Akita University, Japan)

Program Committee:

Francine Blanchet-Sadri (Greensboro)
Victor Bovdi (Al Ain)
Erzsébet Csuhaj-Varjú (Budapest, co-chair)
Jürgen Dassow (Magdeburg)
Pál Dömösi (Debrecen, co-chair)
Henning Fernau (Trier)
Zoltán Fülüp (Szeged)
Viliam Geffert (Kosice)
Marian Gheorghe (Bradford)
Oscar H. Ibarra (Santa Barbara, CA)
Szabolcs Iván (Szeged)
Galina Jirasková (Kosice)
Juhani Karhumäki (Turku)
Alica Kelemenová (Opava)
Miklós Krész (Szeged)
Martin Kutrib (Giessen)
György Maróti (Veszprm)
Alexander Meduna (Brno)
Victor Mitrana (Bucharest, Madrid)
Andreas Maletti (Stuttgart)
Benedek Nagy (Famagusta)
Chrystopher Nehaniv (Hertfordshire)
Friedrich Otto (Kassel)
Gheorghe Păun (Bucharest)
Giovanni Pighizzini (Milano)
Agustín Riscos-Núñez (Sevilla)

Jeffrey Shallit (Waterloo, Canada)
Petr Sosík (Opava)
Bianca Truthe (Giessen)
György Vaszil (Debrecen, co-chair)
Laurent Vuillon (Chambéry)
Claudio Zandron (Milano)

Proceedings, special issue

The proceedings will be published in the EPTCS series (Electronic Proceedings in Theoretical Computer Science, `http://about.eptcs.org/`).

A special issue of the International Journal of Foundations of Computer Science containing expanded versions of selected papers will be published after the conference.

Submissions:

Submissions to AFL2017 must not exceed 15 pages (in EPTCS style and including bibliography). If the authors believe that more details are essential to substantiate the main claims, they may include a clearly marked appendix that will be read at the discretion of the program committee. Simultaneous submission of papers to any other conference with published proceedings or submitting previously published papers is not allowed. Only electronic submissions in PDF format are accepted.

Information about the submission procedure is available on the conference web page (`http://www.inf.unideb.hu/afl2017`).

Organizing Committee:

Géza Horváth (Debrecen, co-chair)
Ernoné Kása (Debrecen)
Gábor Kolonits (Budapest)
Katalin Anna Lázár (Budapest)
György Vaszil (Debrecen, co-chair)
Ildikó Vecsei (Debrecen)

Contact:

`afl2017@inf.unideb.hu`
`csuhaj@inf.elte.hu` (Erzsébet Csuhaj-Varjú)
`domosi@unideb.hu` (Pál Dömösi)
`vaszil.gyorgy@inf.unideb.hu` (György Vaszil)

Reports on MC Conferences/Meetings

Report about 15th BWMC

The 15th Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from January 31 to February 3, 2017, hosted by the Research Group on Natural Computing (RGNC) from the Department of Computer Science and Artificial Intelligence of Universidad de Sevilla.

In the style of previous editions in this series, the organization of the meeting was focused on the cooperation among the participants. The program included sessions for not-so-formal talks, where participants were supposed to present ideas, open problems and research topics in a "provocative" way, in order to trigger interactions. On the other hand, joint-work sessions were scheduled in the afternoons, so that the participants have collaborated in a friendly atmosphere. The program was available at `www.gcn.us.es/15bwmc_program`

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

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

- BWMC 2012: *International Journal of Computer Mathematics* – volume 99, number 4, 2013;
- BWMC 2013: *Romanian Journal of Information Science and Technology*, vol. 17, nr. 1, 2014;
- BWMC 2014: *Fundamenta Informaticae*, volume 134, numbers 1-2, 2014;
- BWMC 2015: *Natural Computing* – volume 15, issue 4, 2016 (some papers from ACMC 2015 were also selected for this special issue);
- BWMC 2016: *Theoretical Computer Science* – to appear.

***

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

1. José Antonio Andreu Guzmán, Universidad de Sevilla (Spain)
   andreuguzman36@gmail.com
2. Lucie Ciencialová, Silesian University (Opava, Czech Republic)
   ciecilka@gmail.com
3. Rudolf Freund, Technological University of Vienna (Austria)
   rudifreund@gmx.at
4. Carmen Graciani, Universidad de Sevilla (Spain)
   cgdiaz@us.es
5. Miguel A. Gutiérrez-Naranjo, Universidad de Sevilla (Spain)
   magutier@us.es
6. Sergiu Ivanov, Université Paris-Est Créteil (France)
   sivanov@colimite.fr
7. Gábor Kolonits, Eötvös Loránd University (Hungary)
   kolonits.gabor@gmail.com
8. Alberto Leporati, University of Milan-Bicocca (Italy)
   leporati@disco.unimib.it
9. Francisco J. Macías García, Universidad de Sevilla (Spain)
   franmaciassevilla@gmail.com
10. Luca Manzoni, University of Milan-Bicocca (Italy)
    luca.manzoni@disco.unimib.it
11. Miguel A. Martínez del Amor, Universidad de Sevilla (Spain)
    mdelamor@us.es
12. David Orellana Martín, Universidad de Sevilla (Spain)
    dorellana@us.es
13. Mario de J. Pérez-Jiménez, Universidad de Sevilla (Spain) and Academia Europaea
    marper@us.es
14. Antonio Enrico Porreca, University of MilanBicocca (Italy)
    porreca@disco.unimib.it
15. Agustín Riscos-Núñez, Universidad de Sevilla (Spain)
    ariscosn@us.es

16. Daniel Rodríguez Chavarría, Universidad de Sevilla (Spain)
danrodcha@gmail.com
17. Álvaro Romero-Jiménez, Universidad de Sevilla (Spain)
romero.alvaro@us.es
18. Eduardo Sánchez-Karhunen, Universidad de Sevilla (Spain)
sanchek@gmail.com
19. Jose María Sempere Luna, Politechnical University of Valencia (Spain)
jsempere@dsic.upv.es
20. Luis Valencia Cabrera, Universidad de Sevilla (Spain)
lvalencia@us.es
21. Gyorgy Vaszil, University of Debrecen (Hungary)
vaszil.gyorgy@inf.unideb.hu
22. Lian Ye, Chongqing University (China)
ylredleaf@cqu.edu.cn

The meeting was partially supported by Universidad de Sevilla, more precisely by Department of Computer Science and Artificial Intelligence and *VI Plan Propio, Vicerrectorado de Investigación de la Universidad de Sevilla.*

Research Group on Natural Computing
Universidad de Sevilla
(March 2017)

A Summary of the Second China Workshop on Membrane Computing (CWMC 2017)

Gexiang Zhang[1,2], Qiang Yang[1], Linqiang Pan[3]

[1] Robotics Research Center, Key Laboratory of Fluid and Power Machinery of Ministry of Education, Xihua University, Chengdu, 610039, China
[2] School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031, China
[3] School of Automation, Huazhong University of Science and Technology, Wuhan 430074, China

The Second China Workshop on Membrane Computing (CWMC 2017) was successfully held at Xihua University, Chengdu, China, from 6-9 April 2017, with the full support of Xihua University. The aim of this workshop is to provide a friendly, flexible, and collaborative platform to Chinese researchers in the area of membrane computing and related topics. Hopefully, this workshop has strengthened the collaborations among the participants and has inspired some new ideas. More than 60 participants, including professors, lecturers, Ph.D. students and master students from about 17 universities, attended the conference. The invited speaker, professor Linqiang Pan, presented his group's work. More than 20 researchers presented their recent results and open problems on both theoretical and application aspects of membrane computing.

A feature of CWMC 2017 is that a French researcher, professor Sergey Verlan, joined this workshop, and an impressive session on how to prepare and write a professional paper at the level of international journal standard was organized.

Invited Talk 1 – Biocomputing and Its Applications

Linqiang Pan, Professor
Huazhong University of Science and Technology, Wuhan, China
E-mail: lqpan@mail.hust.edu.cn

Professor Pan introduced his research group from the aspects of backgrounds, research topics, work done in the recent years, achievements, and future researches.

Professor Pan's group focuses on bio-inspired computing and related nanotechnology, such as membrane computing and DNA nanotechnology.

The research backgrounds mainly include the following points:

- The semiconductor industry will soon abandon its pursuit of Moore's Law.

- The development of biotechnology and nanotechnology gives technical support to information processing based on biological materials.
- Biocomputing provides a platform for non-deterministic computation.
- The leading scientists initiated the ideas on biocomputing and push its development.

The research work on experiments includes:

- Information coding units were developed.
- Multiple input logic gates were developed.
- Biocomputing units were cascaded.
- DNA nanotubes with large diameter were developed.

The research work on theory includes:

- Time-free computing models were developed.
- Computing models with small computing resources were constructed.

The research work on simulation software includes:

- The simulation software P-Lingua has been used to simulate several biocomputing models.

The research achievements include:

- The research team has published 76 SCI-indexed papers in the past five years, where 16 papers were published in *IEEE Trans.*, 11 papers in *Theoretical Computer Science*, and one paper in *PNAS*.
- The research team has been involved in several research projects, including a key project of National Natural Science Foundation of China, a key international joint research program, 15 general programs from National Natural Science Foundation of China, and 4 research programs for young researchers from National Natural Science Foundation of China.
- The research team was awarded with Natural Science Award (first class) by Ministry of Education of China and Natural Science Award (first class) by Hubei province.

In the future, the group will focus on the following research topics:

- Computing models that are close to the biological implementation will be developed.
- Biological implementation of evolutionary algorithms such as simulated annealing algorithms and genetic algorithms will be investigated.
- The application of biocomputing will be considered such as modeling biological systems and using biocomputing models as a cell doctor.

Open Problems 1 – Open Problems on the Complexity of Tissue P Systems

Linqiang Pan, Professor

Huazhong University of Science and Technology, Wuhan, China
E-mail: lqpan@mail.hust.edu.cn

Inspired by the biological phenomena, cell division or cell separation is introduced into tissue P systems in order to obtain exponential workspace in polynomial time. It has been shown that tissue P systems with cell division or cell separation can solve **NP**-complete problems in a polynomial time, and that the known upper bound to the complexity class that confluent tissue P systems with cell division or cell separation characterize is **PSPACE**.

Some open problems about cell- or tissue-like P systems with cell division or cell separation remain to be explored.

- Can cell-like P systems with symport/antiport rules and membrane division characterize **PSPACE**?
- Can **PSPACE** be reached by tissue P systems with division or separation rules if the systems are allowed to be non-confluent?
- Can confluence and determinism in other variants of P systems, in particular, P systems with active membranes, characterize the same complexity classes?
- Is there a resource bound that identifies distinct complexity classes for tissue P systems with cell division and cell separation?
- Minimize the length of communication rules. It is known that a maximum length of 2 across all communication rules in tissue P systems with cell division suffices for solving **NP**-complete problems in polynomial time; furthermore, a length of 3 across all communication rules in tissue P systems with cell separation suffices for solving **NP**-complete problems (assuming that $\mathbf{P} \neq \mathbf{NP}$). It is unknown whether these values are enough to solve $P^{\#P}$ problems.

Open Problems 2 – Open Problems about Spiking Neural P Systems with Polarizations

Tingfang Wu, Ph.D. student
Huazhong University of Science and Technology, China
tfwu@hust.edu.cn

In spiking neural (SN, for short) P systems, the application of a rule is controlled by checking the number of spikes in the neuron against a regular expression associated with the rule. However, it is an **NP**-complete problem to decide whether a natural number is in the length set of the language associated with a regular expression. Therefore, it is a rather natural idea to avoid using regular expressions and consider easy ways to determine the applicability of rules. A new mechanism for controlling the applicability of rules by using electrical charges (i.e., positive, neutral, and negative charges) associated with neurons is introduced. The resulting systems, called SN P systems with polarizations (in short, PSN P systems), are proved to be Turing universal.

Some research topics about PSN P systems remain to be explored:

- A universal PSN P system with 164 neurons was constructed. By using the extended rules and adding the feature of delay, the improvement about the

number of neurons is obtained. It is meaningful and challenging to significantly decrease the number of neurons in the universal PSN P system.
- Besides the synchronized (sequential in each neuron) mode considered in PSN P systems, it deserves to consider PSN P systems working in the non-synchronized mode, the exhaustive mode, or the sequential mode.
- In the proof of the universality of PSN P systems, three charges are used. Is it possible to decrease the number of charges, but without the loss of the computation power?

Open Problems 3 – Open Problems about Tissue-like P Systems with Promoters

Bosheng Song, Postdoctor
Huazhong University of Science and Technology, China
Email: boshengsong@hust.edu.cn

Tissue P systems with promoters (TP P systems, for short) are distributed parallel computing models, and several theoretical results have already been obtained. It is proved that TP P systems using only antiport rules of length 2 or using only symport rules of length 1 are able to compute only finite sets of non-negative integers. Moreover, TP P systems with one cell and using antiport rules of length 2 and symport rules of length 1 or only using symport rules of length 2 are Turing universal. When cell division is considered in TP P systems (TPD P systems, for short), we present a uniform solution to the `SAT` problem by TPD P systems using only antiport rules of length 2.

The open problems are as follows:

- The computation power of TP P systems working in the maximally parallel way has been investigated. What is the computation power of TP P systems by using rules in the flat maximally parallel way?
- Small universal P systems have been studied widely. It remains open how to construct small universal tissue P systems with promoters.
- The length of symport/antiport rules is an essential parameter for the descriptional complexity of P systems. It is interesting to investigate tissue P systems with promoters and cell separation from a computational complexity perspective.

Open Problems 4 – Open Problems on Numerical P Systems with Production Thresholds

Zhiqiang Zhang, Postdoctor
Huazhong University of Science and Technology, China
Email: zhiqiangzhang@hust.edu.cn

In numerical P systems with production thresholds, each program is associated with a constant (called threshold). The production function value can be distributed only when it is not smaller (the lower-threshold case) or not greater (the upper-threshold case) than its associated threshold.

- What is the computation power of numerical P systems with thresholds working in the one-parallel mode?
- It remains open whether or not the number of thresholds used in our proofs can be decremented.
- What are other ways to use the thresholds, such as taking the same threshold for the whole system or for the same membrane?
- What is the computation power of numerical P systems with thresholds when mixing the ways to use the thresholds in the lower and upper ways; or considering the threshold in a strict way?

Presentation 1 – Power network fault location with distributed generation

This presentation discussed the use of SN P systems to realize power network fault location with distributed generation. This work focuses on the transformation from state quantities to electrical quantities.

Presentation 2 – The application of energy control in multi-microgrids system with distributed membrane systems

Based on a distributed membrane system, a novel feasible idea might be proposed to deal with the energy control in a multi-microgrids system.

Presentation 3 – The application of membrane computing in operation and control strategies of micro power grid

The operation and control model of micro power grid is constructed by SN P systems.

Presentation 4 – Automatic design of SN P systems

An evolutionary way is introduced to implement automatic design of an SN P system. This is the first attempt to consider this topic. This study starts from the simplified formulation of automatic design of an SN P system by considering a redundant evolution rule set.

Presentation 5 – Selection of single-phase-to-ground fault lines using SN P systems

An information fusion method is developed by using a fuzzy reasoning SN P system to decide the single-phase-to-ground fault line in a power system. The fault features are first extracted from the zero sequence component. The line fault measure was introduced based on the fault features. Rough set theory is used to fuse the features. Fuzzy reasoning SN P systems are used to construct the classifier.

Presentation 6 – The application of membrane controller in Quadrotors

A parallel membrane controller is being considered to implement on FPGA.

Presentation 7 – Implementation of KNN categorization algorithm based on membrane computing

An improved KNN categorization algorithm is proposed based on P systems.

Presentation 8 – Membrane clustering

Based on the spectral graph theory and membrane computing theory, a method for clustering dataset is proposed.

Presentation 9 – Some ideas about hardware implementation of membrane systems

This research focuses on the representation of membrane configuration, the implementation of maximally parallelism computing and the execution of non-deterministic selection of evolution rules on hardware circuits. This is a very challenging topic.

Presentation 10 – Mathematical model of membrane algorithms

This work focuses on the design of membrane algorithms by combing a P system and genetic algorithms or ant colony optimization. Traffic network layout optimization problems are considered in the experiments.

The group photo of CWMC 2017

Fig. 1. Group photo from China Workshop on Membrane Computing

Miscellanea

About the Limits of (Bio)Informatics With Some Illustrations from DNA and Membrane Computing*

Gheorghe Păun

Romanian Academy, Bucharest, Romania
gpaun@us.es, curteadelaarges@gmail.com

Summary. This is an informal, mainly autobiographical discussion about a series of limits-frontiers-borderlines appearing in computer science, often addressed in natural computing, in particular, in bio-computing. One only mentions such limits dealing with the competence and the performance of computing models and of existing computers, as well as other "impossibility theorems" known in the literature.

Keywords: Turing computability, Turing-Church thesis, Gandy theorem, P versus NP, DNA computing, membrane computing, Conrad theorems, no free lunch theorems

1 Introduction

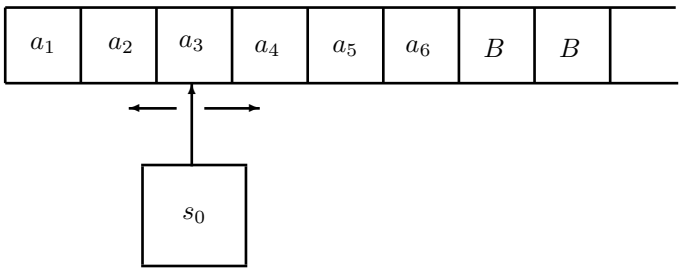
The topic announced in the title above is a subject for a book, here I will only mention a few ideas and references. The semantics itself of the notions *limit, frontier, borderline* should be clarified, but I will rely on the intuitive meaning of them, as well as on the examples considered below. An explicit and systematic discussion about limits is not very usual in computer science, although the limits are present everywhere and most researches, in particular, the unconventional models of computation, are directly motivated by such limits met by the existing models and computers. Here I only mention some of these limits, together with attempts to overpass them, promises, achievements and criticisms, with new limits appearing in the new frameworks - with a few illustrations from the two areas

* Informal presentation dedicated to acad. Mircea Maliţa, on the occasion of his 90th birthday, April 2017.

of bio-inspired computation I have worked in the last (more than) two decades, DNA and membrane computing. Three classes of limits are considered: concerning the computing power (*competence*), the computing efficiency (*performance*), and impossibility theorems (like Conrad theorems). Of course, there are many other similar limits/frontiers (just two examples: the borderline between decidable and undecidable, in automata and language theory, the borderline between universality and non-universality in membrane computing), but I will not touch them here. The discussion is informal, but the references provided allow the reader to start exploring deeper the domain.

2 The Turing "Barrier"

The general framework of what follows is that of Turing computability, based on the notion of what is now called *a Turing machine*, the answer Alan Turing proposed in his 1936 PhD thesis (*Systems of Logic Based on Ordinals*, written in Princeton under the guidance of Alonzo Church) to David Hilbert question (from 1928) concerning "what is mechanically computable". Turing abstracted the way a human being computes until reaching the "minimalistic" device in the figure below: a tape infinite to the right, with a read-write head able to read a cell of the tape, under the control of a state from a finite set of states, to change the contents of the cell and the state and to move to the left or to the right. Although so simple at the first sight, this device was proved to be able to simulate all computing models proposed before (by Post, Church, Kleene, Gödel) and became the standard definition of the notion of an *algorithm*.



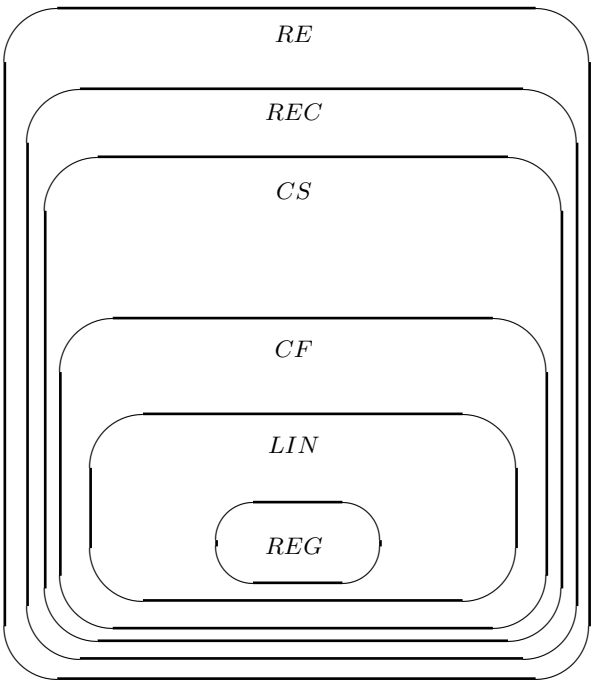
In his thesis, Turing not only introduced his machine, but he has also provided the first example of a problem which cannot be solved by it (namely, *the halting problem:* given an arbitrary Turing machine, is there another Turing machine which can say whether the arbitrary machine halts when starting with an arbitrarily given input on its tape?) and, furthermore, proved the existence of a *universal Turing machine*, a fixed one able to simulate any particular Turing machine as soon as a code of the particular one is placed on the tape of the universal machine (this

suggested to John von Neumann the architecture of the programmable computers he has constructed at the beginning of 1940). See also (Turing, 1936).

The next figure suggests a "map" of computability, in the form of *the Chomsky hierarchy*, with the class of Turing computable functions (languages, numbers, decidable problems) denoted by RE (from "recursively enumerable"). The smallest class, *REG*, denotes the family of regular languages, corresponding to the computing power of *finite automata*, the most restricted class of Turing machines.

These are the two "poles of computability": according to *the Turing-Church Thesis, everything that is algorithmically computable can be computed by a Turing machine*, that is, *RE* is the largest family of computable functions/numbers/languages.

Actually, the thesis has several versions, and there are several papers discussing it. I mention here only (Doyle, 2002).



Can we pass beyond the "Turing barrier"? This possibility was named *hypercomputing*, and there are many proposals, more than one dozen, of how we can "compute the uncomputable". Here are only three references: (Ord, 2002), (Ord, 2006), (Syropoulos, 2008).

Interesting enough, Turing itself proposed a way to compute more than his machine, by considering *Turing machines with oracles*. In the meantime, many other ideas were examined: coupled Turing machines, networks of Turing machines, ora-

cles via (quantum) randomness, accelerated machines/processes, infinite time Turing machines, neural networks with real numbers as weights, Turing machines with an infinite input, and so on and so forth. The book (Syropoulos, 2008) provides details and references.

However, all these are considered by Martin Davis tricks - something uncomputable is introduced in the machine and then the machine is proved, e.g., to solve the halting problem for Turing machines, which already Turing proved that this is a unsolvable problem: (Davis, 2004), (Davis, 2006).

As expected, these criticisms raised counterarguments from the people involved in hypercomputing, see, e.g., (Sundar and Bringsjord, 2011).

Anyway, biocomputing brings new ideas and motivations in this area. One of the ideas is suggested by the strategy used in 1994 by Leonard Adleman, in his history making experiment of solving the Hamiltonian Path Problem (known to be **NP**-complete, hence intractable for the Turing machine) in linear time by using DNA molecules, (Adleman, 1994).

This was a great achievement, a *demo* that DNA can be used as a support for computing (Hartmanis, 1994), the start of DNA computing as a branch of natural computing. Actually, at the theoretical level, DNA computing started in 1987, when T. Head introduced his *splicing operation*, as a language theory model of the recombinant behavior of DNA molecules, (Head, 1987).

In the Adleman experiment one starts by producing all possible paths in a graph (in the form of DNA molecules), then one filters out molecules which do not encode Hamiltonian paths. Otherwise stated, in terms of languages, one starts from a given set of strings and one removes strings until reaching the solution, that is, one removes the complement of the solution. This is the idea of *computing by carving*, proposed in (Păun, 1997).

In this way, we can compute languages outside RE, because the family RE is not closed under complement.

Actually, a "reasonable" way to "carve" is proposed in the mentioned paper: one starts from a regular language (hence easy to compute) and we repeatedly remove a sequence of regular languages which are linked to each other in the following way: the first language is given, the next one is obtained from the first one by means of a sequential translation (the simplest kind of transducers), and so on.

Formulated as a theorem, the conclusion is that *a language is computable in this way (by carving) if and only if it can be written as the complement of a recursively enumerable language.*

Also membrane computing can suggest hypercomputing ideas.

I only mention that membrane computing is a branch of natural computing initiated in (Păun, 1998), with a rapid development, (Păun, 2002). It abstracts computing models (usually called P systems), from the architecture and the functioning of the living cells. A comprehensive presentation of the domain can be found in (Păun, Rozenberg, and Salomaa, 2010), with news available at the domain webpage http://ppage.psystems.eu.

The cell membranes have two main roles in the cells: to enclose "protected reactors", with a specific biochemistry, and to facilitate the collision of reactants.

This means that smaller the space, faster the reactions. In the style of accelerated machines (which perform the first step of a computation in one time unit, the second one in half of a time unit, and so on, decreasing by two the external time needed to perform each next step), we can assume that the reactions in a membrane inside another membrane are twice faster than in the upper membrane. Generalizing this assumption, we get a way to solve the halting problem: (Calude and Păun, 2003).

In both cases (carving and accelerating by creating membranes inside membranes) we pass beyond the family RE, the biology motivates/supports the ideas, nothing infinite or uncomputable was introduced in the model itself, but still Martin Davis is right: in both cases the process should be infinite (the carving and the hierarchy of constructed membranes). If we stop computing after a finite number of steps, then we remain inside RE.

I end this section by mentioning an important result concerning the Turing barrier, namely, *Gandy's Theorem*, from (Gandy, 1980).

Robin Gandy was a student and collaborator of Turing who tried to get an abstract description of a "computable device". He coined four principles (*The finiteness of description, The principle of limitation of hierarchy, The principle of unique reassembly, The principle of local causality*), formulated in algebraic terms, and proved that *whatever can be calculated by a device satisfying principles I – IV is Turing computable.*

These principles are general enough to support Martin Davis opinion about hypercomputability, but they also suggest ways to go beyond Turing; for instance, the last principle suggests that a machine able of transmitting instantaneously signals at an arbitrary distance (on its tape) could be able of hypercomputation.

3 Feasible versus Unfeasible (or P versus NP)

The competence (computing power) is important, in particular, the equivalence with Turing machines is important practically (this brings "for free" the universality, hence the programmability of the computing device), but in applications it is still more important to know the resources (space and, mainly, time) needed in order to compute something. This is the motivation behind the powerful theory of computational complexity developed in computer science, a framework where problems which can be solved in a time polynomial with respect to the size of the input are considered *tractable* and the problems requesting an exponential time (for the known algorithms, but solvable in polynomial time in a non-deterministic way) are considered *intractable*. The two classes of complexity are denoted by $\mathbf{P}$ and $\mathbf{NP}$, respectively. The inclusion of $\mathbf{P}$ in $\mathbf{NP}$ is obvious, the question whether or not $\mathbf{P} = \mathbf{NP}$ is probably the most important open problem of computer science. This is also confirmed by the fact that this problem is the first one in the

list of seven "Millennium Problems" compiled by the Clay Mathematics Institute (`www.claymath.org`) in 2000, which provides a prize of one million dollars for a solution.

Such a solution can be of various forms: strict inclusion, equality proved in a nonconstructive manner, constructive equality with huge coefficients and exponents of polynomials, constructive equality with reasonable coefficients - the consequences for practical computing increases in this order.

However, the expectation is that **N** is not equal to **NP** and then, with this assumption in mind, one looks for ways to pass over the "feasibility barrier"; imitating the term hypercomputing, the term *fypercomputing* was proposed in (Păun, 2012) for such an achievement.

As mentioned in the previous section, DNA computing started with such a promise, illustrated by Adleman's experiment. The strategy is based on trading space for time, making use of the fact that DNA molecules are very efficient data supports, of a nanometric size. However, exponentially many DNA molecules, as in Adleman's experiment, do not really help, as soon observed, (Hartmanis, 1995).

This is the case in today DNA computing: many successful computing experiments were reported, but all of them are dealing with toy problems. Scaling-up to the level of problems of a practical size looks unfeasible (this could request huge amounts of DNA). Further details (as well as many theoretical developments, based, e.g., on the Head splicing operation) can be found in the monograph (Păun, Rozenberg, and Salomaa, 1998).

Many theoretical ways to solve **NP**-complete problems in a polynomial time, hence to obtain fypercomputations, are also provided by membrane computing, again trading space by time. In this framework, the exponential space is obtained by means of biologically inspired operations, such as membrane division, membrane creation, string replication. Another interesting idea is to start with arbitrarily large pre-computed resources, without containing "too much" information (not to be possible to hide the solution of the problem there and then to claim that it is obtained through a simulated computation), to introduce the problem in this given pre-computed space, then to activate as much space as necessary, and solve the problem (this is the way the brain and the liver are functioning).

Like in the case of DNA computing, also in this case the Hartmanis criticism is valid, as we need an exponential working space (created during the computation, through bio-like operations).

Interesting enough, the creation of new, exponentially many membranes (or strings) cannot be avoided. This is stated by the so-called *Milano Theorem* from (Zandron, 2001).

From a practical point of view, another strategy is much more useful: looking not for exact optimal solutions, but for approximate solutions, known to be good with a specified probability. This is the strategy of a very developed area of natural computing, namely *evolutionary computing*. It contains a large number of classes of algorithms, usually based on a search in the space of solutions, making use of the brute force of the existing computers, with the search conducted in a way in-

spired from biology. There are many classes of such approaches: genetic algorithms, immune computing, ant colony algorithms, bee colony algorithms, swarm computing, water flowing algorithms, cultural algorithms, cuckoo algorithms, strawberry algorithms, firefly algorithm, etc. etc.

These algorithms have a lot of practical applications, but also this approach has a drastic limitation, first provided in (Wolpert and Macready, 1997): informally formulated, the no free lunch theorem says that *all approximative algorithms are equally good* (one can also read "equally bad"), for each of them there are problems for which the provided solutions are bad.

4 Further Limits

Computer science contains also further limits. A typical one is pointed out by *Conrad Theorems* – see references in (Conrad, 1988).

The main theorem says that *a computing system cannot at the same time have high programmability, high computational efficiency, and high evolutionary adaptability.* It is rather possible that further similar "impossibility theorems" can be proved (if too many conditions are simultaneously imposed, then no computing model exists which satisfies all of them).

Such theorems are probably possible also in more general frameworks, for instance, in biological modeling, where the user (the biologist) asks for many features (understandability, scalability, adequacy to reality, relevance), at the same time with the computer scientist (easy programmability, efficiency) - and it is possible that no model exists to fulfil all requests.

Still more general: some classic experts of artificial intelligence, such as R. Brooks and J. McCarthy, have even estimated that it is possible that the current mathematics itself is not sufficiently developed for modeling such subtle notions like life and intelligence, and a new stage of mathematics is necessary for making significant progresses in this area.

About such limits of mathematics has warned us also acad. Mircea Maliţa, already in *Aurul cenuşiu* (*The Grey Gold*), vol. II, Ed. Dacia, Cluj-Napoca, 1972. And, the limits of mathematics extend also to (theoretical) computer science.

References

1. L.M. Adleman: Molecular computation of solutions to combinatorial problems. *Science*, 226 (Nov. 1994), 1021–1024.
2. C. Calude and Gh. Păun: Bio-steps beyond Turing, CDMTCS Research Report 226, Univ. of Auckland (November 2003), and *BioSystems*, 77 (2004), 175–194.
3. M. Conrad: The price of programmability. In *The Universal Turing Machine: A Half-Century Survey*, R. Herken, ed., Kammerer and Unverzagt, Hamburg, 1988, 285–307.
4. M. Davis: The myth of hypercomputation. In *Alan Turing: Life and Legacy of a Great Thinker*, C. Teuscher, ed., Springer, 2004, 195–211.

5. M. Davis: Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178, 1 (2006), 4–7.
6. J. Doyle: What is Church's Thesis? An outline. *Minds and Machines*, 12 (2002), 519–520.
7. R. Gandy: Church's Thesis and principles for mechanisms. In *The Kleene Symposium*, J. Barwise, H.J. Keisler, and K. Kunen, eds., North-Holland, 1980, 123–148.
8. J. Hartmanis: About the nature of computer science. *Bulletin of the EATCS*, 53 (June 1994), 170–190.
9. J. Hartmanis: On the weight of computation. *Bulletin of the EATCS*, 55 (February 1995), 136–138.
10. T. Head: Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49 (1987), 737–759.
11. T. Ord: *Hypercomputation: Computing More Than the Turing Machine*. Honours Thesis, Dept. of CS, Univ. of Melbourne, September 2002.
12. T. Ord: The many forms of hypercomputation. *Applied Mathematics and Computation*, 178 (2006), 143–153.
13. Gh. Păun: (DNA) Computing by carving. Technical Report CTS-97-17, Center for Theoretical Study at Charles Univ. and the Academy of Sciences of the Czech Republic, Prague, 1997, and *Soft Computing*, 3, 1 (1999), 30–36.
14. Gh. Păun: Computing with membranes. Turku Center for Computer Science-TUCS Report No 208, 1998 (www.tucs.fi), and *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
15. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
16. Gh. Păun: Towards "fypercomputations" (in membrane computing). In *Languages Alive. Essays Dedicated to Jrgen Dassow on the Occasion of His 65 Birthday*, H. Bordihn, M. Kutrib, and B. Truthe, eds., LNCS 7300, Springer, Berlin, 2012, 207–221.
17. Gh. Păun, G. Rozenberg, and A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer, Berlin, 1998; Springer, Tokyo 1999; Tsinghua Univ. Press, Beijing, 2004; Mir, Moskow, 2005.
18. Gh. Păun, G. Rozenberg, and A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford Univ. Press, 2010.
19. N. Sundar G. and S. Bringsjord: The myth of "The myth of hypercomputation". *Proc. of Combined P&C2011/HyperNet 11*, May 2011, 185–196.
20. A. Syropoulos: *Hypercomputation: Computing Beyond the Church-Turing Barrier*. Springer, Berlin, 2008.
21. A.M. Turing: On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, Ser. 2, 42 (1936), 230–265; a correction, 43 (1936), 544-546.
22. D.H. Wolpert and W.G. Macready: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1, 1 (1997), 67–82.
23. C. Zandron: *A Model for Molecular Computing: Membrane Systems*. PhD Thesis, University of Milano-Bicocca, Milano, Italy, 2001.

Contents of Previous Volumes

June 2016 Volume

Letter from the President 7

IMCS Matters 11

– Structure of IMCS 11

– Constitution of IMCS 17

– Report from the Treasurer 23

News from MC Research Groups 25

– Gexiang Zhang: NICSG: Nature-Inspired Computation and Smart Grid Lab in China 25

– Thomas Hinze: Interlocal Research Group on Molecular and Membrane Computing at Brandenburg University of Technology (Cottbus) and Friedrich Schiller University (Jena) 29

– Research Group of Theoretical Computer at Science Silesian University in Opava 33

– Research Group on Natural Computing – Universidad de Sevilla 37

– Vincenzo Manca: The Research Group in Bioinformatics at the University of Verona 41

Tutorials, Surveys, Bibliographies 47

– Marian Gheorghe: A Survey of Membrane Computing Results Published Between CMC14 and CMC15 47

– Zhiqiang Zhang, Tingfang Wu: A Bibliography of Numerical P Systems 59

– Linqiang Pan, Tingfang Wu, Zhiqiang Zhang: A Bibliography of Spiking Neural P Systems 63

Open Problems, Inquiries, Answers 79

– Gheorghe Păun, Tingfang Wu, Zhiqiang Zhang: Open Problems, Research Topics, Recent Results on Numerical and Spiking Neural P Systems (The "Curtea de Argeş 2015 Series") 79

– Artiom Alhazov, Rudolf Freund, Sergiu Ivanov: Spiking Neural P Systems with Polarizations – Two Polarizations Are Sufficient for Universality 97

Software News 105

– Andrei George Florea, Cătălin Buiu: Lulu, an open-source P colony/P swarm simulator..... 105

Abstracts of PhD Theses 107

– Luis Felipe Macías-Ramos: *Developing Efficient Simulators for Cell Machines* 107

– Pradeep Isawasan: *Variants of Array-Rewriting P Systems for Generating Picture Arrays* 112

– Francis George C. Cabarle: *Computations in Spiking Neural P Systems: Simulations and Structural Plasticity* 116

Reports on MC Conferences/Meetings 119

– Report about 14th BWMC 119

– H.N. Adorna, F.G.C. Cabarle, N.H.S. Hernandez, R.A.B. Juayong: A Report about the Membrane Computing Seminar (together with PCSC2016), March 2016, at Puerto Princesa, Palawan, Philippines 125

– A Summary of 2016 China Workshop on Membrane Computing (CWMC 2016) 127

Miscellanea 135

Gheorghe Păun: Obituary Solomon Marcus (1925–2016) 135

December 2016 Volume

Letter from the President 9

IMCS Matters 11

– Structure of IMCS 11

– Constitution of IMCS 17

News from MC Research Groups 23

– The Research Group on Modeling, Simulation and Verification of Biological Systems at the University of Pisa 23

– Research Group on Bio-Inspired Computing at Huazhong University of Science and Technology in China 27

– Hong Peng, Jun Wang: RGMCA: Research Group on Membrane Computing and Applications at Xihua University, Chengdu 31

– José M. Sempere: The Spanish Thematic Network on Biomolecular and Biocellular Computing 37

– Formal Methods Laboratory (FML) in Iaşi 43

Bibliographies 47

– Marian Gheorghe: Kernel P Systems Bibliography 47

– PhD Theses in Membrane Computing 49

– Membrane Computing, 16th International Conference, CMC 2015, Valencia, Spain, August 17-21, 2015, Revised Selected Papers, LNCS 9504, Springer-Verlag, Berlin, 2015 55

– Ludek Cienciala: Membrane Agents – Book Summary 61

– Andrew Adamatzky, ed.: Advances in Unconventional Computing. Vol. I: Theory, Vol. 2: Prototypes, Models and Algorithms. Springer, 2016 63

Tutorials, Surveys 69

– Luis Valencia-Cabrera, David Orellana-Martín, Agustín Riscos-Núñez, Mario J. Pérez-Jiménez: Complexity Perspectives on Minimal Cooperation in Cell-like Membrane Systems 69

– Artiom Alhazov, Rudolf Freund, Sergiu Ivanov: Polymorphic P Systems: A Survey 79

– Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo, Hong Peng: Some Notes on Membrane Computing and Image Processing 103

– Lucie Ciencialová, Erzsébet Csuhaj-Varjú, Luděk Cienciala, Petr Sosík: P Colonies 129

Technical Notes, Open Problems, Inquiries, Answers 157

– Vincenzo Manca: Multiset Generalization of Balanced Chemical Reactions 157

– Matteo Cavaliere, Alvaro Sanchez: Evolutionary Resilience of Membrane Computations 159

– Gheorghe Păun, José M. Sempere: Families of Languages Associated with SN P Systems: Preliminary Ideas, Open Problems 161

– Marian Gheorghe: Membrane Systems Analysis 165

– Alan Mehlenbacher: P System for Two-Level Economic Exchange with Investment 167

– Linqiang Pan, Gheorghe Păun, Gexiang Zhang: SN P Systems with Communication on Request 179

Abstracts of PhD Theses 195

– Christian Bodenstein: *Theoretical Analysis of Cyclic Processes in Biology in the Context of* Ca^{2+} *Oscillations, Circadian Rhythms and Synthetic Oscillators* 195

– Zhiqiang Zhang: *Research on the Computational Power of Numerical P Systems* 200

– Lea Weber: *Implementation of an Artificial Evolution for Optimal Placement of Processing Units on a Freely Configurable Two-dimensional Grid Map* 204

– Sergiu Ivanov: *On the Power and Universality of Biologically-Inspired Models of Computation* 207

– Ciprian Dragomir: *Formal Verification of P Systems* 210

– Tao Wang: *Spiking Neural P Systems and Their Applications in Fault Diagnosis of Electric Power Systems* 212

Calls for Participation to MC Conferences/Meetings 217

– Call for Participation Fifteenth Brainstorming Week on Membrane Computing 217

– CMC 18, 24-28 July 2017, Bradford, UK. First Call for Papers 219

– Call for Papers WMC at UCNC 2017, Fayetteville, USA 223

Reports on MC Conferences/Meetings 225

– Alberto Leporati, Claudio Zandron: Report on CMC17 The Seventeenth Conference on Membrane Computing 225

– Linqiang Pan, Gexiang Zhang, Ravie Chandren Muniyandi, Bosheng Song: A Summary of The 5th Asian Conference on Membrane Computing (ACMC 2016) 229

– Marian Gheorghe, Savas Konur: Workshop on Membrane Computing at the International Conference on Unconventional Computation and Natural Computation, Manchester, UK, July 11-15 2016..... 235

– Svetlana Cojocaru, Alexandru Colesnicov, Ludmila Malahov: Workshop on Unconventional Computing Systems in commemoration of Yurii Rogozhin Chişinău, Moldova, November 11, 2016 237

Miscellanea 241

Gheorghe Păun: Some Wonders of a Bio-Computer-Scientist 241