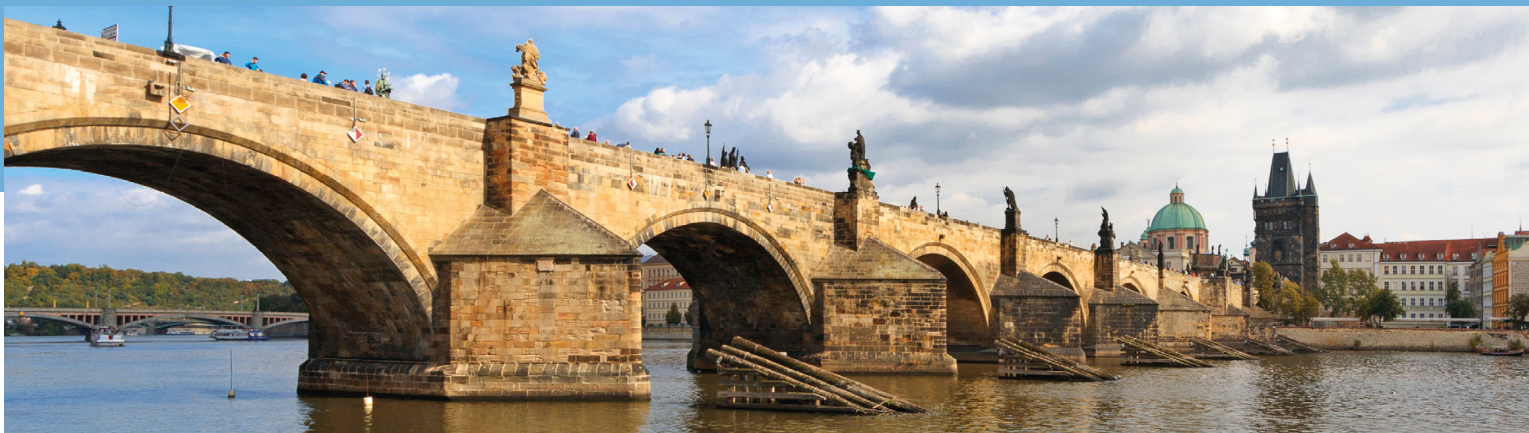


Marian Gheorghe
Petr Sosík
Šárka Vavrečková (Eds.)

The 15th International Conference on Membrane Computing



Proceedings

CMC15

**Prague, Czech Republic,
August 20–22, 2014**

Institute of Computer Science
Faculty of Philosophy and Science
Silesian University in Opava

Marian Gheorghe
Petr Sosík
Šárka Vavrečková (Eds.)

15th International Conference on Membrane Computing

CMC15, Prague, Czech Republic, August 20–22, 2014

Proceedings

Institute of Computer Science
Faculty of Philosophy and Science
Silesian University in Opava

**15th International Conference
on Membrane Computing**

Conference page: <http://cmc15.slu.cz/>

Institute of Computer Science
Faculty of Philosophy and Science in Opava
Silesian University in Opava
Bezručovo náměstí 13
746 01 Opava
Czech Republic

Web page: <http://www.slu.cz/fpf/en/institutes/the-institute-of-computer-science>

Editors: © Marian Gheorghe (Sheffield, UK)
Petr Sosík (Opava, Czech Republic)
Šárka Vavrečková (Opava, Czech Republic)

Copyright: © Institute of Computer Science, Silesian University in Opava
Authors are fully responsible for the content of their papers.

Cover Design: © Karel Melecký

Published in Opava, July 2014

ISBN 978-80-7510-036-8

Preface

This volume contains a collection of papers presented at CMC15, the 15th International Conference on Membrane Computing, Prague, Czech Republic, during 20th to 22nd August, 2014 (<http://cmc15.slu.cz/>).

The CMC series was initiated by Gheorghe Păun as the Workshop on Multi-set Processing in the year 2000. Then two workshops on Membrane Computing were organized in Curtea de Argeș, Romania, in 2001 and 2002. A selection of papers from these three meetings was published as volumes 2235 and 2597 of the Lecture Notes in Computer Science series, and as a special issue of *Fundamenta Informaticae* (volume 49, numbers 1–3, 2002). The next six workshops were organized in Tarragona, Spain (in July 2003), Milan, Italy (in June 2004), Vienna, Austria (in July 2005), Leiden, The Netherlands (in July 2006), Thessaloniki, Greece (in June 2007), and Edinburgh, UK (in July 2008), with the proceedings published in Lecture Notes in Computer Science as volumes 2933, 3365, 3850, 4361, 4860, and 5391, respectively. The 10th workshop returned to Curtea de Argeș in August 2009 (LNCS volume 5957).

From the year 2010, the series of meetings on membrane computing continued as the Conference on Membrane Computing, held in Jena, Germany (LNCS volume 6501), Fontainebleau, France (LNCS volume 7184), Budapest, Hungary (LNCS volume 7762) and finally Chisinau, Moldova (LNCS volume 8340). Nowadays a Steering Committee takes care of the continuation of the CMC series which is organized under the auspices of the European Molecular Computing Consortium (EMCC). A regional version of CMC, the Asian Conference on Membrane Computing, ACMC, started with 2012 edition in Wuhan, China and continued with Chengdu, China (2013) and Coimbatore, India (2014).

CMC15 is organized by the Institute of Computer Science of the Faculty of Philosophy and Science, Silesian University in Opava, in collaboration with the Action M Agency, Prague. A special session is dedicated to the memory of Prof. Yurii Rogozhin, the head organizer of CMC14, a world-class mathematician and computer scientist and, last but not least, a dear friend of many participants to the CMC series.

The Program Committee of CMC15 invited lectures from Claudio Zandron (Italy), Erzsébet Csuhaj-Varjú (Hungary), Luděk Cienčiala (Czech Republic), Mario J. Pérez Jiménez (Spain), and Jiří Wiedermann (Czech Republic).

In addition to extended abstracts of the invited talks, this volume contains 22 papers of regular talks and two extended abstracts presented at the Conference. Each paper was subject to at least two referee reports.

The editors warmly thank the Program Committee, the invited speakers, the authors of the papers, the reviewers, and all the participants for their contributions to the success of CMC15.

August 2014

Marian Gheorghe
Petr Sosík
Šárka Vavrečková

The Steering Committee of the CMC series consists of

Gabriel Ciobanu (Iași, Romania)
Erzsébet Csuhaj-Varjú (Budapest, Hungary)
Rudolf Freund (Vienna, Austria)
Marian Gheorghe (Sheffield, UK) – chair
Vincenzo Manca (Verona, Italy)
Maurice Margenstern (Metz, France)
Giancarlo Mauri (Milan, Italy)
Gheorghe Păun (Bucharest, Romania and Seville, Spain)
Mario J. Pérez-Jiménez (Seville, Spain)
Linqiang Pan (Wuhan, China)
Petr Sosík (Opava, Czech Republic)
Sergey Verlan (Paris, France)

The Organizing Committee consists of

Petr Sosík – co-chair
Petr Čermák – co-chair
Luděk Ciencala
Miroslav Langer
Šárka Vavrečková
Štěpánka Tůmová
Hana Černínová

The Programme Committee of the CMC consists of

Artiom Alhazov (Chişinău, Moldova)
Luděk Ciencala (Opava, Czech Republic)
Gabriel Ciobanu (Iaşi, Romania)
Erzsébet Csuhaj-Varjú (Budapest, Hungary)
Giuditta Franco (Verona, Italy)
Rudolf Freund (Vienna, Austria)
Marian Gheorghe (Sheffield, UK) – co-chair
Thomas Hinze (Jena, Germany)
Florentin Ipate (Bucharest, Romania)
Shankara Narayanan Krishna (Bombay, India)
Alberto Leporati (Milan, Italy)
Vincenzo Manca (Verona, Italy)
Maurice Margenstern (Metz, France)
Giancarlo Mauri (Milan, Italy)
Radu Nicolescu (Auckland, New Zealand)
Linqiang Pan (Wuhan, China)
Gheorghe Păun (Bucharest, Romania and Seville, Spain)
Mario J. Pérez-Jiménez (Seville, Spain)
Dario Pescini (Milan, Italy)
Agustín Riscos-Núñez (Seville, Spain)
Yurii Rogozhin (Chişinău, Moldova)
Petr Sosík (Opava, Czech Republic) – co-chair
György Vaszil (Debrecen, Hungary)
Sergey Verlan (Paris, France)
Claudio Zandron (Milan, Italy)
Gexiang Zhang (Chengdu, Sichuan, China)

Contents

Preface	iii
Invited Papers	
From P colonies to 2D P colonies and to simulations of multiagent systems <i>Luděk Cienciala</i>	3
P systems: A Formal Approach to Social Networks	7
<i>Erzsébet Csuhaj-Varjú, Marian Gheorghe, György Vaszil</i>	
A bioinspired computing approach to model complex systems	11
<i>Mario J. Pérez-Jiménez</i>	
Inconspicuous Appeal of Amorphous Computing Systems	15
<i>Jiří Wiedermann</i>	
P Systems with Active Membranes Working in Sublinear Space	19
<i>Claudio Zandron, Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca</i>	
Regular Papers	
Membrane Computing Inspired Approach for Executing Scientific Workflow in the Cloud	25
<i>Tanveer Ahmed, Rohit Verma, Abhishek Srivastava</i>	
P Systems with Anti-Matter	41
<i>Artiom Alhazov, Bogdan Aman, Rudolf Freund</i>	
Priorities, Promoters and Inhibitors in Deterministic Non-Cooperative P Systems	61
<i>Artiom Alhazov, Rudolf Freund</i>	
P Systems with Toxic Objects	71
<i>Artiom Alhazov, Rudolf Freund</i>	
Length P Systems with a Lonesome Traveler	99
<i>Artiom Alhazov, Rudolf Freund, Sergiu Ivanov</i>	

Promoters and Inhibitors in Purely Catalytic P Systems	117
<i>Artiom Alhazov, Rudolf Freund, Sergey Verlan</i>	
A Short Note on Red-Green P Automata	129
<i>Bogdan Aman, Erzsébet Csuhaaj-Varjú, Rudolf Freund</i>	
Extended Simulation and Verification Platform for Kernel P Systems	135
<i>Mehmet E. Bakir, Florentin Ipate, Savas Konur, Laurentiu Mierla, Ionut Niculescu</i>	
Notes on Spiking Neural P Systems with Structural Plasticity Computing Morphisms	153
<i>Francis George Cabarle, Henry Adorna</i>	
The Abilities of P Colony Based Models in Robot Control	155
<i>Luděk Cienciala, Lucie Ciencialová, Miroslav Langer, Michal Perdek</i>	
Probabilistic Guarded P systems, a New Formal Modelling Framework ...	169
<i>Manuel García-Quismondo, Miguel á. Martínez-Del-Amor, Mario J. Pérez-Jiménez</i>	
Solving the ST-Connectivity Problem with Pure Membrane Computing Techniques	191
<i>Zolt Gazdag and Miguel A. Gutiérrez-Naranjo</i>	
Simulating Turing Machines with Polarizationless P Systems with Active Membranes	205
<i>Zolt Gazdag, Miguel A. Gutiérrez-Naranjo, Gábor Kolonits</i>	
Categorised Counting Mediated by Blotting Membrane Systems for Particle-based Data Mining and Numerical Algorithms	217
<i>Thomas Hinze, Konrad Grützmann, Benny Höckner, Peter Sauer, Sikander Hayat</i>	
Polymorphic P Systems with Non-cooperative Rules and No Ingredients ..	235
<i>Sergiu Ivanov</i>	
Simulating Elementary Active Membranes, with an Application to the P Conjecture	251
<i>Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, Claudio Zandron</i>	
Spiking Neural P Systems with Cooperating Rules	267
<i>Venkata Padmavati Metta, Srinivasan Raghuraman, Kamala Krithivasan</i>	

On the Dynamics of P Systems Based on Membrane Boundaries	283
<i>Tamás Mihálydeák, Zoltán Ernő Csajbók</i>	
Parallel Thinning with Complex Objects and Actors	287
<i>Radu Nicolescu</i>	
Causal Nets for Geometrical Gandy–Păun–Rozenberg Machines	313
<i>Adam Obtulowicz</i>	
P System Computational Model as Framework for Hybrid (Membrane-Quantum) Computations	333
<i>Yurii Rogozhin, Artiom Alhazov, Liudmila Burtseva, Svetlana Cojocaru, Alexander Colesnicov, Ludmila Malahov</i>	
Expressing Workflow and Workflow Enactment using P Systems	345
<i>Rohit Verma, Tanveer Ahmed, Abhishek Srivastava</i>	
Fault Diagnosis Models for Electric Locomotive Systems Based on Fuzzy Reasoning Spiking Neural P Systems	361
<i>Tao Wang, Gexiang Zhang, Mario J. Pérez-Jiménez</i>	
Graph Cluster Analysis by Lattice based P System with Conditional Rules	375
<i>Jie Xue, Xiyu Liu</i>	
Author Index	377

Invited Papers

From P colonies to 2D P colonies and to simulations of multiagent systems

Luděk Cienciala

Institute of Computer Science
and

Research Institute of the IT4Innovations Centre of Excellence,
Silesian University in Opava, Czech Republic
ludek.cienciala@fpf.slu.cz

Abstract

P colonies are a class of abstract computation devices based on one-membrane agents acting in a shared environment. They belongs to a family of models inspired by biology and biochemistry of cells called P systems introduced in [11] by Gheorghe Păun in 2000.

Each agent is represented by a collection of objects embedded in a membrane and by a set of programs for processing these objects. The number of objects placed inside each agent is unchangeable and it is called the capacity of P colony. The computational abilities in particular depend on the capacity of P colony, on the number of agents and on the type of processing rules in the programs.

The rules used in programs are rewriting $a \rightarrow b$, communication $c \leftrightarrow d$ or checking r_1/r_2 . Using rewriting rule agent evolves object a to object b . Both objects have to be placed inside this agent. Using rewriting rule agent change its state. If the communication rule $c \leftrightarrow d$ is applicable, the object c must be contained inside the agent and there is at least one copy of object d in the environment. By applying communication rule the object c moves to the environment and one object d comes to the agent. We can say, that agent took information d from the environment and left information c in the environment. The checking rule is not really new type of rules but checking rule can be obtained by putting together two rules of previous types. This provides a pair of rules and the order determines a priority among them.

Computational power of such a kind of devices with or without using checking rules has been a point of interest of lots of research papers (e.g. [4, 8, 7]) and it was shown, that they are computationally complete.

The environment is a communication channel for agents and storage place for objects. It plays strategic role in synchronization of works of single agents during computation. The environment has become the most changing / extending part of P colonies.

In the eco-P colonies ([3, 1]) the static environment was replaced by the evolving one using 0L-scheme. The input tape was add to P colony in the model called Pcol automaton ([2]). The last model derived from P colony uses environment

resembling cellular automata and it is called 2D P colony ([5]). The environment is changed to a form of a 2D grid of square cells. The agents are located in this grid and their view is limited to the cells that immediately surround them. Based on the contents of these cells, the agents decide their future locations. This formal model seems to be suitable for e.g. simulations of artificial and natural multiagent systems.

In the presentation we describe the individual models, compare them from different viewpoints and we outline the development of models from the original model of P colonies to 2D P colonies.

Remark 1.

This work was partially supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by project OPVK no. CZ.1.07/2.2.00/28.0014.

References

1. Cienciala, L., Ciencialová, L.: Eco-P colonies, In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A. (eds.) Pre-Proceedings of the 10th Workshop on Membrane Computing, Curtea de Arges, Romania, pp. 201–209 (2009)
2. Ciencialová, L., Cienciala, L., Csuhaĵ–Varjú, E., Vaszil, Gy.: PCol Automata: Recognizing Strings with P Colonies. Report of Eight Brainstorming week on membrane computing, Sevilla, Spain (2010)
3. Ciencialová, L., Csuhaĵ–Varjú, E., Kelemenová, A., Vaszil, Gy.: On Very Simple P Colonies. Proceeding of The Seventh Brainstorming Week on Membrane Computing, vol. I, Sevilla, Spain, pp. 97–108 (2009)
4. Ciencialová, L., Cienciala, L., Kelemenová, A.: *On the number of agents in P colonies*, In G. Eleftherakis, P. Kefalas, and G. Paun (eds.), *Proceedings of the 8th Workshop on Membrane Computing (WMC'07)*, June 25-28, Thessaloniki, Greece, 2007, pp. 227–242.
5. Cienciala, L., Ciencialová, L., Perdek, M.: 2D P colonies. In Csuhaĵ–Varjú et al. (eds.). CMC 2012, Springer, LNCS 7762, 2013, pp. 161–172.
6. Csuhaĵ–Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, G.: *Cells in environment: P colonies*, Multiple-valued Logic and Soft Computing, 12, 3-4, 2006, pp. 201–215.
7. Csuhaĵ–Varjú, E., Margenstern, M., Vaszil, G.: *P Colonies with a bounded number of cells and programs*. Pre-Proceedings of the 7th Workshop on Membrane Computing (H. J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), Leiden, The Netherlands, 2006, pp. 311–322.
8. Freund, R., Oswald, M.: *P colonies working in the maximally parallel and in the sequential mode*. Pre-Proceedings of the 1st International Workshop on Theory and Application of P Systems (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, pp. 49–56.
9. Kelemen, J., Kelemenová, A.: *On P colonies, a biochemically inspired model of computation*. Proc. of the 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, pp. 40–56.
10. Kelemen, J., Kelemenová, A., Păun, Gh.: *Preview of P colonies: A biochemically inspired computing model*. Workshop and Tutorial Proceedings, Ninth International

- Conference on the Simulation and Synthesis of Living Systems, ALIFE IX (M. Bedau et al., eds.) Boston, Mass., 2004, pp. 82–86.
11. Păun, Gh.: *Computing with membranes*. Journal of Computer and System Sciences 61, 2000, pp. 108–143.
 12. Păun, Gh.: *Membrane computing: An introduction*. Springer-Verlag, Berlin, 2002.
 13. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.
 14. Torrents, B.: *Geosimulation*. John Wiley & Sons, 2004.
 15. P systems web page: <http://psystems.disco.unimib.it>

P systems: A Formal Approach to Social Networks (Abstract) *

Erzsébet Csuhaj-Varjú¹, Marian Gheorghe² and György Vaszil³

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

² Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello, Sheffield S1 4DP, United Kingdom
`m.gheorghe@sheffield.ac.uk`

³ Department of Computer Science, Faculty of Informatics
University of Debrecen, PO Box 12, 4010 Debrecen, Hungary
`vaszil.gyorgy@inf.unideb.hu`

One of the major challenges for membrane computing is to find applications of the obtained theoretical results in different scientific areas. Concepts and methods in P systems theory have been so far successfully employed in solving various problems in computer science and in modelling several biological phenomena, but except applications in linguistics and natural language processing, only a limited amount of attention has been paid to utilization of membrane systems in social sciences.

One of the few steps in this direction was made in [12] where P systems (membrane systems) were proposed to model social networks, an area of contemporary computer science and practice which is in the focus of interest. Purely syntactic models have already been used for describing communities of agents interacting with each other and with their dynamically changing environment. Examples are the framework of eco-grammar systems, from the theory of grammar systems, launched in [10], and networks of (parallel) language processors describing populations of agents by rewriting systems in [14, 8]. Multi-agent systems in terms of formal language theory and membrane computing were discussed in [5], [6], [4], and [17].

Roughly speaking, social networks are communities of individuals forming a communication network based on some social phenomena. When formalizing these networks, their special features, as interpersonal relationships between individuals, are expected to appear in the syntactic model. In various formalisms related to the study of social phenomena, these relationships are defined as information-carrying connections [16]. Two basic types of them are *strong* and *weak ties*. Weak ties are responsible for the embeddedness and structure of social networks and for the communication within these systems [16]. There are other

* Research supported in part by the Hungarian Scientific Research Fund, “OTKA”, project K75952.

measures that characterize connections between nodes (agents). *Centrality* gives an indication of the social power of a node and the strength of its connections. It relies on other measures, *betweenness*, *closeness*, *degree*. Betweenness is the degree of connectivity between a node and the nodes that have a significant number of neighbours (direct connections). Closeness measures the distance between a node and all the other nodes in the network: it counts the number of connections. For more details on these concepts and for some other measures of connections existing in social networks the reader is advised to consult [25].

P systems, especially tissue P systems, can also be considered as collections of agents (individuals) communicating with each other: the compartments or nodes with the multisets of objects may represent the individuals and the rules of the system describe the communication/interaction between the components. In the case of population P systems, see [7], the established communication links may dynamically change. Notice that the objects can also be considered as information pieces, in this case a node represents a loosely organized community of agents.

In [12] new classes of P systems capturing communication aspects in social networks were introduced and various research topics related to connections between P systems theory and the theory of social interactions and networks were initiated. They are called *population P systems governed by communication* (pcgP systems, for short). In this case, in addition to the multisets of standard objects which are called cellular objects, so-called communication objects are present in the network. The transition takes place by rewriting and communication of the cellular objects and recording the performed communication. The transitions are governed by communication, i.e., rules can be performed only if some predicates on the multisets of communication symbols associated to the links are satisfied. Whenever communication takes place, the number of communication symbols associated to the link increases.

It is easy to see that the model provides various possibilities to study the behaviour of the nodes: *ordinary* or *popular* nodes - those that host individuals and allow communication between them; *new-born* nodes - those that are dynamically created and linked to the existing network; *non-visible* or *extinct* nodes - the nodes that are no longer connected to the network or have disappeared; nodes with one way communication, only allowing information to go into, *blackholes* or allowing only to exit from, *whiteholes*. Some of these aspects have been already discussed in membrane computing: for example, population P systems allow nodes to be dynamically connected and disconnected [7]. We can also take into account connections between nodes and look at the *volume of communication* - the amount of (new) information generated or sent-received by various nodes or groups of nodes; *frequency of communicated messages* - the number of communication steps related to the evolution (computation) steps; *communication motifs* - patterns of communication identified throughout the network evolution.

In [12] we have focused on communication in these networks. In order to characterize it and the fact that this might evolve in time by either increas-

ing or decreasing its value, we introduced some sort of symbols that act in this respect, called complementary communication symbols. The customary (or positive) symbols strengthen a connection, whereas the complementary (negative) ones weaken it.

The idea of complementary alphabets is not new in the field of natural computing. It is a core idea of *DNA computing*, and the concept of related notions have been discussed in membrane computing as well [1], [2], [3], [20], [19].

In [12] some further concepts regarding some specific types of pgcP systems have also been defined, as well as presented some preliminary results for *deterministic* and *non-cooperative* pgcP systems, based on tools of Lindenmayer systems (D0L systems). Among others, we described the growth of the communication volume, the frequency, and the intensity of communication on the links.

In this talk we will discuss pgcP systems with different types of predicates for communication, and demonstrate how the choice of predicates affects the volume and the intensity of communication, the type of communication motifs in the network. We also will characterize other *concepts and measures* from social networks like *leaders and clusters emergence*. Dynamical restructuring the links, including break of links will also be topic of our discussion.

References

1. A. Alhazov, B. Aman, R. Freund, Gh. Păun. Matter and Anti-Matter in Membrane Systems. Brainstorming Week on Membrane Computing, Sevilla, 2014.
2. B. Aman, G. Ciobanu. Turing completeness using three mobile membranes. In *Unconventional Computing 2009*, Lecture Notes in Computer Science, 5715, 42–55, 2009.
3. B. Aman, G. Ciobanu. Mutual mobile membranes systems with surface objects. In *7-th Brainstorming Week of Membrane Computing*, 29–39, 2009.
4. G. Bel-Enguix. A Multi-agent Model for Simulating the Impact of Social Structure in Linguistic Convergence. In *ICAART(2)* (J. Filipe et. al, Eds.), INSTICC Press, 367–372, 2009.
5. G. Bel-Enguix, M. A. Grando, M. D. Jiménez López. A Grammatical Framework for Modelling Multi-Agent Dialogues. In *PRIMA 2006* (Z.-Z. Shi, R. Sadananda, Eds.), LNAI 4088, Springer Verlag, Berlin Heidelberg, 10–21, 2009.
6. G. Bel-Enguix, M. D. Jiménez López. Membranes as Multi-agent Systems: an Application for Dialogue Modelling. In *IFIP PPAI 2006* (J.K. Debenham, Ed.), Springer, 31–40, 2006.
7. F. Bernardini, M. Gheorghe. Population P systems. *Intern J of Universal Comp Sci*, 10, 509–539, 2004.
8. E. Csuhaj-Varjú. Networks of Language Processors. *EATCS Bulletin* 63, 120–134, 1997.
9. E. Csuhaj-Varjú. Computing by networks of Watson-Crick D0L systems. In *Proc. Algebraic Systems, Formal Languages and Computation* (M. Ito, Ed.) RIMS Kokyuroku 1166, August 2000, Research Institute for Mathematical Sciences, Kyoto University, Kyoto, 43–51, 2000.

10. E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun. Eco-Grammar Systems: A Grammatical Framework for Studying Lifelike Interactions. *Artificial Life*, 3(3), 1–28, 1997.
11. E. Csuhaj-Varjú, V. Mitran. Evolutionary systems: a language generating device inspired by evolving communities of cells. *Acta Informatica*, 36(11), 913–926, 2000.
12. E. Csuhaj-Varjú, M. Gheorghe, M. Oswald, Gy. Vaszil. P systems for Social Networks. In *BWMC11* (M.A. Martnez-del-Amor, Gh. Pün, I. Pérez-Hurtado, F.J. Romero-Campero, L. Valencia-Cabrera, eds.), Fénix Editora, 113–124, 2011.
13. E. Csuhaj-Varjú, A. Salomaa. Networks of Watson-Crick *D0L* systems. In *Words, Languages & Combinatorics III. Proceedings of the International Colloquium, Kyoto, Japan, March 14-21, 2000.* (M. Ito, T. Imaoka, Eds.), World Scientific Publishing Co., Singapore, 134–149, 2003.
14. E. Csuhaj-Varjú, A. Salomaa. Networks of Parallel Language Processors. In *New Trends in Formal Languages. Control, Cooperation, and Combinatorics* (Gh. Păun, A. Salomaa, Eds.), LNCS 1218, Springer Verlag, Berlin Heidelberg, 299–318, 1997.
15. E. Csuhaj-Varjú, G. Vaszil. P automata or purely communicating accepting P systems. In *Membrane Computing*, LNCS 2579, Springer Verlag, Berlin Heidelberg, 219–233, 2003.
16. M.D. Granovetter. The Impact of Social Structures on Economic Development. *Journal of Economic Perspectives*, 19, 33–50, 2004.
17. M. D. Jiménez López. Agents in Formal Language Theory: An Overview. In *Highlights in Practical Applications of Agents and Multiagent Systems. 9th International Conference on Practical Applications of Agents and Multiagent Systems* (J. Bajo Pérez et. al, Eds.) Advances in Intelligent and Soft Computing 89, Springer, 283–290, 2011.
18. M. Minsky. *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
19. Gh. Păun. *Membrane computing. An Introduction*. Springer, 2002.
20. L. Pan, G Păun. Spiking neural P systems with anti-spikes, *Int J Computers Comms Control*, 4, 273–282, 2009.
21. Gh. Păun. Computing with Membranes. *J. of Comput. Syst. Sci.*, 61, 108–143, 2000.
22. Gh. Păun, G. Rozenberg, A. Salomaa. *DNA Computing - New Computing Paradigms*. Springer Verlag, 1998.
23. G. Rozenberg, A. Salomaa. (Eds). *Handbook of Formal Languages I-III*. Springer, 1997.
24. Gh. Păun, G. Rozenberg, A. Salomaa. (Eds). *The Handbook of Membrane Computing*. Oxford University Press, 2009.
25. S. Wasserman, K. Faust. *Social Networks Analysis: Methods and Applications*. Cambridge University Press, 1994.

A bioinspired computing approach to model complex systems

Mario J. Pérez–Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Computational modelling is the process of representing real world problems in mathematical terms in an attempt to find solutions to their associated complex systems. Modelling is an integral part of many scientific disciplines and lies behind great human achievements and developments [11]. Scientists regularly use abstractions with the aim to describe and understand the reality they are examining. A good model should have four properties: relevance, computability, understandability and extensibility [14].

One of the main objectives of any model is to provide a predictive capability, that is, the possibility to make conjectures of plausible hypothesis related to the dynamics of the observed phenomenon in different scenarios. It is important to notice that different models might be able to produce similar spatio-temporal behaviour but they are distinguished by the different experiments they suggest.

Usually, cellular systems and population biology often depend on many variables of the observed behaviours. Since they define the dynamics of the system, variables must satisfy some conditions, which can be referred as the invariants of the related behaviour. Some of these invariants can be expressed by rules and can be obtained by realizing experiments, while others cannot be measured in the lab or are very expensive to estimate. Therefore, before simulations can be performed in order to make predictions, we need to *calibrate* our model by obtaining estimations for missing variables and validate it against the expected behaviour of the system. This requires comparing trial results or simulations obtained by using the model with real data coming from the lab or with data generated using *reliable* methods. This stage involves an iterative process in which a candidate set of parameters is proposed, some simulations are generated and, on the basis of some metric of closeness to the desired behaviour, a new set of parameters is tested. In some cases, design of the model has to be reconsidered.

Nowadays ordinary/partial differential equations (ODEs/PDEs) constitute the most widely used approach in modelling complex systems. For example, they have been successfully used to model kinetics of conventional macroscopic chemical reactions. Nevertheless, in some cases such as molecular interaction networks in cellular systems, any model described by means of a system of ODEs/PDEs is based on two assumptions: (a) cells are assumed to be well stirred and homogeneous volumes so that concentrations do not change with respect to space; and (b) chemical concentrations vary continuously over time in a deterministic

way. This assumption is valid if the number of molecules specified in the reaction volume (the cell or the subcellular compartment) are sufficiently large and reactions are fast. A sufficient large number of molecules is considered to be of the order of, at least, thousands of molecules.

Membrane Computing is an emergent branch of Natural Computing introduced by G. Paun. This new computing paradigm starts from the assumption that processes taking place in the compartmental structure of a living cell can be interpreted as computations. In contrast to differential equations, P systems explicitly correspond to the discrete character of the components of an ecosystem and use rewriting rules on multisets of objects which represent the variables of the system. The inherent stochasticity, external noise and uncertainty in cellular systems is captured by using stochastic or probabilistic strategies. While the stochastic approach is usually applied to model *micro*-level systems (such as bacteria colonies, signalling pathways, etc.), the probabilistic one is normally used for *macro*-level modelling (of real ecosystems, for example).

A multienvironment P system can be viewed as a finite set of environments and a finite set of P systems, such that: (a) the links between environments are given by the arcs taken from a directed graph; (b) each P system has the same working alphabet, the same membrane structure and the same evolution rules; and (c) each environment contains several P systems, where each evolution rule has associated a computable function and each one of them has an initial multiset which depends on the environment; and (d) there is a finite set of rules among the environments. Furthermore, inside the environments, only objects from a distinguished alphabet can exist.

In the stochastic approach the following holds: (a) the computable functions associated with the rules of the P systems are *propensities*, obtained from the kinetic constants [12] (these rules depend on time but not on the environment); and (b) initially, the P systems are randomly distributed among the environments of the system. These kind of P systems are called *multicompartmental P systems*. Dynamics of these systems is captured by either the multicompartmental Gillespie's algorithm [12] or the deterministic waiting time algorithm [2]. Some practical examples of multicompartmental P systems applications for modelling cellular systems are: signalling pathways (Epidermal Growth Factor Receptor [13], FAS-induced apoptosis [2]), gene expression control in Lac Operon [16] and quorum sensing in *Vibrio Fischeri* [15].

In the probabilistic approach the following holds: (a) the total number of P systems $\Pi_{k,j}$ is equal to the number of environments: each environment contains one P system; (b) functions associated with rules of P systems are *probability functions* verifying some conditions; (c) rules among the environments have associated some *probability functions* verifying some conditions. The dynamics of these systems is captured by *ad hoc* algorithms such as Binomial block based simulation algorithm [6], DNDP (Direct Non Deterministic distribution with Probabilities) algorithm [9] and DCBA (Direct distribution based on Consistent Blocks Algorithm) algorithm [10], among others. These kind of P systems are called *population dynamics P systems* (PDP systems) [7]. Some practical ex-

amples of PDP systems applications for modelling real ecosystems are: bearded vultures in the Pyrenees (NE Spain) [3], avian scavengers in the Pyrenean and Prepyrenean mountains of Catalonia, NE Spain [4], Pyrenean Chamois (*Rupicapra p. pyrenaica*) in the Catalan Pyrenees [5], zebra mussel in the dam of Ribarroja (NE region of Spain) [1], and pandemics [8].

Acknowledgements

The author acknowledges the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, co-financed by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10, 1 (2011), 39-53.
2. S. Cheruku, A. Păun, F.J. Romero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems. *Proceedings of the First International Conference on Bio-Inspired Computing: Theory and Applications*, Wuhan, China, September, 18-22, 2006.
3. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, D. Sanuy, A. Margalida. Modeling ecosystems using P systems: The bearded vulture, a case study. *Lecture Notes in Computer Science*, 5391 (2009), 137-156
4. M.A. Colomer, A. Margalida, D. Sanuy, M.J. Pérez-Jiménez. A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. *Ecological modelling*, 222, 1 (2011), 33-47.
5. M.A. Colomer, S. Lavín, I. Marco, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, E. Serrano, L. Valencia-Cabrera. Modeling population growth of Pyrenean Chamois (*Rupicapra p. pyrenaica*) by using P systems. *Lecture Notes in Computer Science*, 6501 (2011), 144-159.
6. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos. Comparing simulation algorithms for multienvironment probabilistic P system over a standard virtual ecosystem. *Natural Computing*, 11 (2012), 369-379.
7. M.A. Colomer, A. Margalida, M.J. Pérez-Jiménez. Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools. *PLOS ONE*, 8 (4): e60698 (2013) (doi: 10.1371/journal.pone.0060698).
8. M.A. Colomer, M. García-Quismondo, L.F. Macías-Ramos, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, A. Riscos- Núñez, L. Valencia-Cabrera. Membrane systems-based models for specifying population systems. In P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez (eds.) *Applications of Membrane Computing in Systems and Synthetic Biology*. Series: Emergence, Complexity and Computation, Vol. 7, 2014.
9. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos, F. Sanchó. A simulation algorithm for multienvironment probabilistic P systems: A formal verification. *International Journal of Foundations of Computer Science*, 22, 1 (2011), 107-118.

10. M.A. Martínez, I. Pérez, M. García, L.F. Macías, L. Valencia, A. Romero, C. Graciani, A. Riscos, M.A. Colomer, M.J. Pérez-Jiménez. DCBA: Simulating population dynamics P systems with proportional objects distribution. *Lecture Notes in Computer Science*, 7762 (2013), 257-276.
11. L. Nunes de Castro (ed.) *Fundamentals of Natural Algorithms. Basic concept, algorithms and applications*, Chapman and Hall/CRC, Taylor and Francis Group, Boca Raton, FL, 2006.
12. M.J. Pérez-Jiménez, F.J. Romero-Campero. P Systems, a New Computational Modelling Tool for Systems Biology, *Transactions on Computational Systems Biology VI, LNBI 4220* (2006), 176-197.
13. M.J. Pérez-Jiménez, F.J. Romero-Campero. A Study of the Robustness of the EGFR Signalling Cascade using Continuous Membrane Systems. *Lecture Notes in Computer Science*, 3561 (2005), 268 – 278.
14. A. Regev, E. Shapiro. The π -calculus as an abstraction for biomolecular systems. In Gabriel Ciobanu and Grzegorz Rozenberg, editors, *Modelling in Molecular Biology*, Springer Berlin, 2004.
15. F.J. Romero-Campero, M.J. Pérez-Jiménez. A model of the Quorum Sensing system in *Vibrio fischeri* using P systems, *Artificial Life*, 14, 1 (2008), 95–109.
16. F.J. Romero-Campero, M.J. Pérez-Jiménez. Modelling gene expression control using P systems: The Lac Operon, a case study, *BioSystems*, 91, 3 (2008), 438-457.

Inconspicuous Appeal of Amorphous Computing Systems*

(Invited talk)

Jiří Wiedermann

Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. Amorphous computing systems typically consist of myriads of tiny simple processors that are randomly distributed at fixed positions or move randomly in a confined volume. The processors are “embodied” meaning that each of them has its own source of energy, has a “body” equipped with various sensors and communication means and has a computational control part. Initially, the processors have no identifiers and from the technological reasons, in the interest of their maximal simplicity, their computational, communication, sensory and locomotion (if any) parts are reduced to an absolute minimum. The processors communicate wirelessly, e.g., in an airborne medium they communicate via a short-range radio, acoustically or optically and in a waterborne medium via molecular communication. In the extreme cases the computational part of the processors can be simplified down to probabilistic finite state automata or even combinatorial circuits and the system as a whole can still be made universally programmable. From the theoretical point of view the structure and the properties of the amorphous systems qualify them among the simplest (non-uniform) universal computational devices. From the practical viewpoint, once technology will enable a mass production of the required processors a host of new applications so far inaccessible to classical approaches to computing will follow.

Extended abstract: The history of amorphous computing systems began by the end of the twentieth century, mainly as an engineering endeavor (cf. [1], [2], [4], [5], [6], or [13]). Namely, in those days the progress in constructing the micro-electro-mechanical systems (MEMS) has enabled to think of devices integrating a central data processing unit (the microprocessor) and several components that interact with the surroundings such as micro-sensors, wireless communication unit, and the energy source in a small unit. These parts can possibly be complemented by micro-actuators and locomotive means. The resulting device can be viewed as an embodied computational unit. Note that such a unit possesses all the necessary parts characterizing autonomous embodied robots.

MEMS devices generally range in size from 20 micrometres (20×10^{-6} m) to a millimetre (i.e. 0.02 to 1.0 mm). Current ideas about nano-electro-mechanical systems (NEMS) and nano-technology consider such systems at a nano-scale (10^{-9} m).

* This work was partially supported by RVO 67985807 and the GA ČR grant No. P202/10/1333

The driving force behind the respective development has mainly been a vision of huge amounts of the respective “micro-robots” engaged in various application tasks requiring local computation, local sensing, local actions and wireless communication among the randomly distributed units of the system. The joint idea has been that using local communication, the respective devices could self-organize in order to perform a coordinated action none of the elements alone was able to realize.

It is obvious that the resulting system of locally communicating units has no fixed architecture what is reflected by the term “amorphous computing systems”.

Amorphous computing systems could, e.g., be spread from an airplane on the surface over a certain region in order to monitor certain parameters of the environment, such as temperature, humidity, precipitation, or life manifestations (on an other planet) (cf. [3], [5], [6], [12], [13], and [14]). These measurements can be transmitted into a base station that will make the data processing. Such gadgets can also be used for object surveillance. Another application is in medical sciences — the devices can be attached to patients and spread over the hospitals in order to monitor the patients’ movements and life functions. A nano-size device of such kind can even enter human bodies in order to perform genetic manipulations at the cell level, to strengthen the immune system, to heal up injuries, to cure heart or brain strokes, etc. (cf. [7]). Real bacteria represent a template for such systems already existing in nature.

Amorphous computing systems communicating via radio can be seen as an extreme case of wireless sensory networks: not only are the nodes of amorphous systems considered under severe size and cost constraints resulting into corresponding restrictions on resources such as energy, memory, and computational speed. There are additional limitations usually not considered in the domain of wireless sensory networks: the computational and communication hardware in the processors is stripped down to an absolute minimum that seems to be necessary in order to maintain the required functionality and scalability of the network. For instance, in order to allow potentially unbounded scalability of amorphous computing systems their processors initially do not possess any identifiers: they are all alike, they are very simple and theoretically can be seen as finite state automata. In order to maximally simplify the wireless communication mechanism in the nodes of amorphous computing systems no embedded communication software is assumed. There is no synchronicity assumed and the communication among the nodes is “blind” — a sender has no means for discovering the presence of nearby receivers, it cannot learn that its message has been delivered since a receiver node cannot reliably send message acknowledgement to a message sender. The broadcast itself is disturbance-prone — the simultaneously broadcasting nodes jam each other in such a way that no message can be delivered and moreover, the interference in broadcasting cannot be recognized by the processors.

Waterborn amorphous computing systems usually work on different principles than the radio-driven systems since radio waves do not travel well through good electrical conductors like salt water and similar liquids. Therefore, the former systems communicate with the help of the signal molecules that spread around via Brownian motion [16]. In some cases the decisions of nano-machines are based on so-called quorum sensing [18], i.e., on the density of signal molecules in the environment. This calls for a

completely different design of the communication and control mechanisms that has no counterpart in the domain of classical distributed computing.

The inconspicuous appeal of amorphous computing systems consists in their immense variety of forms, in the possibility of their adaptation to particular characteristics of their operational environment, in their extreme simplicity and, last but not least, in their immense applicability to problems that cannot be solved by classical computational means. All these properties are supported by the computational universality of the underlying systems.

So far, the prevailing focus of research in amorphous computing systems has mostly been focused towards engineering or technological aspects of such systems almost completely ignoring theoretical questions related to their computational power and efficiency. Obviously, without knowing their theoretical limits, one cannot have a complete picture of the potential abilities and limitations of such systems. This was the starting point of the project of the present author and his (then) PhD student L. Petru (cf. his PhD thesis [8]) devoted to studies of theoretical issues in amorphous computing initiated in 2004. Since that time various models of amorphous systems have been investigated.

The aim of the present talk is to give a brief overview of the developments within the corresponding research as performed within our amorphous computing research project. In the talk we present the main design ideas behind the respective models, point to the main problems to be solved, indicate their solution, and present the main results. The models will be approached roughly in the order of their increased generality (cf. [19], [20], [21]).

We start with the simplest model of amorphous cellular automata [8] and will continue with more elaborated asynchronous stationary amorphous computing systems [9], [17]. Then we turn our attention towards the so-called flying amorphous computing systems with mobile processors (cf. [10] and [11]). Finally, we describe molecularly communicating nano-machines that orchestrate their activities either by a molecular analogue of radio broadcast [16] or via quorum sensing [18]. Interestingly, in the latter case the nano-machines must be endowed by the self-reproduction ability.

The main result of our investigations is the proof of the computational universality of the amorphous computing systems considered above. This points to the versatility of such systems in various computational or robotic applications (cf. [15], [17]).

We conclude by stressing that the amorphous computing systems offer a radically new concept in information technology that has the potential to revolutionize the way we communicate and exchange information.

References

1. H. Abelson, et al. Amorphous Computing. MIT Artificial Intelligence Laboratory Memo No. 1665, Aug. 1999
2. H. Abelson, D. Allen, D. Coore, Ch. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, R. Weiss. Amorphous Computing. Communications of the ACM, Volume 43, No. 5, pp. 74–82, May 2000
3. D. K. Arvind, K. J. Wong: Speckled Computing – A Disruptive Technology for Network Information Appliances. Proc. IEEE International Symposium on Consumer Electronics (ISCE'04), 2004, pp. 219-223

4. D. Coore: Introduction to Amorphous Computing. Unconventional Programming Paradigms: International Workshop 2004, LNCS Volume 3566, pp. 99–109, Aug. 2005
5. J. M. Kahn, R. H. Katz, K. S. J. Pister. Next century challenges: mobile networking for “Smart Dust”. In: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom '99, ACM, pp. 271–278, Aug. 1999
6. J. M. Kahn, R. H. Katz, K. S. J. Pister. Emerging Challenges: Mobile Networking for Smart Dust. Journal of Communications and Networks, Volume 2, pp. 188–196, 2000
7. Kurzweil, R.: The Singularity is Near. Viking Books, 652 pages, 2005
8. L. Petrů: Universality in Amorphous Computing. PhD Disseration Thesis. Dept. of Math. and Physics, Charles University, Prague, 2009
9. L. Petrů, J. Wiedermann: A Model of an Amorphous Computer and Its Communication Protocol. In: Proc SOFSEM 2007: Theory and Practice of Computer Science. LNCS Volume 4362, Springer, pp. 446–455, July 2007
10. L. Petrů, J. Wiedermann: A Universal Flying Amorphous Computer. In: Proc. Unconventional Computation, 10th International Conference, UC'2011, LNCS, Vol. 6714, 2011, pp. 189-200
11. L. Petrů, J. Wiedermann: A Robust Universal Flying Amorphous Computer. In: C. Calude, R. Freivalds, K. Iwama (Eds.), Jozef Gruska Festschrift, LNCS, 2014, to appear
12. M. J. Sailor, J. R. Link: Smart dust: nanostructured devices in a grain of sand, Chemical Communications, Vol. 11, p. 1375, 2005
13. S. C. Shah, F. H. Chandio, M. Park: Speckled Computing: Evolution and Challenges. Proc. IEEE International Conference on Future Networks, 2009, pp. 181-185
14. B. Warneke, M. Last, B. Liebowitz, K. S. J. Pister: Smart Dust: Ccommunicating with a Cubic-Millimeter Computer. Computer, Volume: 34, Issue: 1, pp. 44–51, Jan. 2001
15. J. Wiedermann, L. Petrů: Computability in Amorphous Structures. In: Proc. CiE 2007, Computation and Logic in the Real World. LNCS Volume 4497, Springer, pp. 781–790, July 2007
16. J. Wiedermann, L. Petrů: Communicating Mobile Nano-Machines and Their Computational Power. In: Third International ICST Conference, NanoNet 2008, Boston, MA, USA, September 14-16, 2008, Revised Selected Papers, LNICST Vol. 3, Part 2, Springer, pp. 123-130, 2009.
17. J. Wiedermann, L. Petrů: On the Universal Computing Power of Amorphous Computing Systems. Theory of Computing Systems 46:4 (2009), 995-1010, www.springerlink.com/content/k2x6266k78274m05/fulltext.pdf
18. Wiedermann, J.: Nanomachine Computing by Quorum Sensing. In: J. Kelemen and A. Kelemenová (Eds.): Paun Festschrift, LNCS 6610, p. 203215, 2011
19. Wiedermann, J.: Amorphous Computing: A Research Agenda for the Near Future. Natural Computing, 2012, Vol. 11, No. 1, p. 59-63.
20. Wiedermann, J.: Computability and Non-computability Issues in Amorphous Computing. In Baeten, J.C.M., Ball, T., de Boer, F.S. (ed.). Theoretical Computer Science. Berlin: Springer, 2012, p. 1-9.
21. Wiedermann, J.: The many forms of amorphous computational systems. In: H. Zenil (Ed.): A Computable Universe. Understanding Computation and Exploring Nature As Computation, p. 243-256, Singapore: World Scientific, 2013

P Systems with Active Membranes Working in Sublinear Space

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

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

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

EXTENDED ABSTRACT

P systems with active membranes have been introduced 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 evolution 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 a polynomial time, and use them in parallel to solve computationally hard problems, such as problems in **NP** or even in **PSPACE**.

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, 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. As a consequence, the size of a configuration of a P system can be defined as the sum of the number of membranes in the current membrane structure and the total number of objects they contain. The space required by a halting computation is the maximum among the sizes of the configurations reached during such a computation. The space complexity of a P system Π , denoted by $|\Pi|$, is the maximum among the sizes of all its computations. Finally, a family of P systems (working on strings from an alphabet Σ) $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$ is said to operate in space $f: \mathbb{N} \rightarrow \mathbb{N}$ if $|\Pi_x| \leq f(|x|)$ for all $x \in \Sigma^*$.

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**. A similar result to characterize the complexity class **EXSPACE** can be obtained by considering exponential space P systems. The results have then been generalized, showing that any Turing machine working in space $\Omega(n)$ can be simulated with a polynomial space overhead.

The talk will focus on P systems working in sublinear space, with a survey on recent research results obtained by our group at the University of Milano-Bicocca. In order to consider sublinear space, we first need to define a corresponding meaningful notion in the framework of membrane systems, inspired by sublinear space definition for Turing machines: we consider P systems having two distinct alphabets, an *INPUT* alphabet and a *WORK* alphabet. The input objects do not contribute (unless they are rewritten) to the size of the configuration, which is defined as the sum of the number of membranes in the current membrane structure and the total number of only the *working* objects they contain. Notice that, since sublinear-space Turing machines are weaker (possibly strictly weaker) than those working in polynomial time, we also need to consider uniformity conditions for the families of P systems that are weaker than the P uniformity usually considered, such as logspace uniformity or even **DLOGTIME** uniformity (usually employed for families of Boolean circuits).

The first natural approach, when considering the use of sublinear space in the framework of membrane systems, is to compare P systems using a logarithmic space with Turing machines using the same amount of space. We first showed that **DLOGTIME**-uniform 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**.

However, this result felt somewhat unsatisfactory: in fact, logarithmic-space Turing machines can only generate a polynomial number of distinct configurations; on the contrary, P systems working in logarithmic space have *exponentially* many potential ones. Indeed, n distinct input objects can, in principle, be arbitrarily partitioned in $\log n$ different regions of the P system. An interesting open question, thus, was the following: is it possible to effectively reach (and distinguish in order to exploit them) all such configurations? The answer is trivially yes when we consider non-confluent P systems, but it was not clear whether or not this could actually be done also in a *confluent* way.

We proved that this was indeed the case, and that it is possible to move the input objects among the membranes and to exploit their position in such a way that harder problems than those in **L** can be solved. In particular, using such a technique, polynomial-space Turing machines can be simulated by means of P systems with active membranes using only logarithmic auxiliary space, thus characterising **PSPACE**. 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.

We considered then P systems using only a *constant* amount of space, and it turned out that, surprisingly, a constant amount of space is sufficient (and trivially necessary) to solve all problems in **PSPACE**. To obtain this result, the input objects are used to store the contents of the tape of the simulated Turing machine, and the construction is based on two main ideas. The first one is that,

since by rewriting more than a constant number of input symbols the amount of space would be non-constant, the only way to use such symbols to store the state of the tape is to track their position inside the membrane structure. The second idea is that it is possible to “read” a subscript of an input object σ_j without rewriting it and by using only a constant number of additional objects and membranes, using the objects to implement a counter and the charges of the membranes to inform other objects that the timer has reached the value j .

This result challenges our intuition of space, formalized in the definition of space complexity for P systems, adopted so far: 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? From an information-theoretic perspective, we may observe that, on an input of size n , the constant number of working objects employed by the simulation actually encode $\Theta(\log n)$ bits of information, since they are taken from an alphabet of polynomial size with respect to n . It may be argued that this amount of information needs a proportional amount of physical storage space. Similarly, the membranes themselves, being identified by a label selected from a set Λ , contain a number of bits of information that is $\Theta(\log \Lambda)$, which must have a physical counterpart.

Following these considerations, we proposed a more accurate estimate of the space required by a configuration of a P system, that takes into account such logarithmic factors. 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.

In fact, the systems used to obtain the result described above, and requiring a constant amount of space according to the former definition, require now a logarithmic amount of space. Furthermore, the space bounds of the simulation of polynomial-space Turing machines by means of logarithmic-space P systems with active membranes, previously described, 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.

We will conclude the talk by presenting a result that highlights the importance of rewriting input symbols for P systems when a sub-logarithmic number of membrane labels is used. We proved that, under this last condition, if we only allow the use of rules that move the input objects in the membrane hierarchy (i.e. send-in, send-out, and dissolution rules), then it is even impossible to correctly distinguish two input strings of the same length (unless the ordering of the symbols is not taken into account), even when no bound on the amount of space is present. In other words, P systems having the limitations just described and accepting (resp., rejecting) a long enough string x also accept (resp., reject) another string obtained by swapping two symbols of x .

Regular Papers

Membrane Computing Inspired Approach for Executing Scientific Workflow in the Cloud

Tanveer Ahmed, Rohit Verma, Miroojin Bakshi, Abhishek Srivastava

Indian Institute of Technology Indore

{phd12120101, phd12110101, cse1200105, asrivastava}@iiti.ac.in

Abstract. The continuous expansion and appreciation of the service oriented architecture is due to the standards of loose-coupling and platform independence. Service-Oriented Architecture is most commonly and effectively realized through web services, and their temporal collaboration commonly referred to as web service composition. In the present scenario, the most popular variant of composition is service orchestration. Orchestration is achieved through a centralized ‘*heavyweight*’ engine, the orchestrating agent, that makes the deployment configuration a massive ‘*choke-point*’. The issue achieves significance when data and compute intensive scientific applications rely on such a centralized scheme. Lately, a lot of research efforts are put in to deploy a scientific application on the cloud, thereby provisioning resources *elastically* at runtime. However, just like the orchestrating agent it is the cloud now that provisions resources and in a similar manner there is an unnecessary burden on the hosted platform. In this paper, we aim at eliminating this central ‘choke’ point by presenting a model inspired from ‘*Membrane Computing*’ that executes a scientific workflow in a decentralized manner. The benefit of this paradigm comes from the natural process of autonomy, where each cell provision resources and execute process-steps on its own. The approach is devised keeping in mind, the feasibility of deployment on a cloud based infrastructure. To validate the model, a prototype is developed and real scientific workflows are executed in-house (within the Intranet). Moreover, the entire prototype is also deployed on a virtualized platform with software defined networking, thereby studying the effects of a low bandwidth environment, and dynamic provisioning of resources.

Keywords: Scientific Workflows, Service Oriented Architecture, Membrane Computing

1 Introduction

Service oriented architecture has now become a multipurpose paradigm executing business processes and helping compute intensive scientific applications [2]. It’s success can largely be attributed to advancements in the field of cloud computing. The technological advancements, notwithstanding, the paradigm relies

on a centralized solution to orchestrate either a business process or data intensive scientific applications as the case may be. The orchestrator, by virtue of its inherent centralized nature is a single point of failure and suffers from the issues of scalability, reliability, fault tolerance, security, and privacy [4].

It is well known that a scientific workflow is resource intensive, whereas a business workflow is control oriented. A scientific workflow is well exemplified by the Large Synoptic Survey Telescope¹ experiment aimed to recursively explore and take images of the sky for a period of 10 years and expected to generate data (300MB/s) in the order of exabytes (both raw and pruned). To rely on a centralized architecture in this context, is a potential processing and communication bottleneck. Moreover, the failure of an orchestrator or the hosted platform (cloud's Infrastructure as a Service), causes a cascaded chain reaction, making the entire hierarchy of deployed services moot. The issue is evident by the failure of Amazon in 2011, 2012, and 2013. A decentralized setup, on the other hand, achieves a reduction in failures and enables quick recovery. Considering the example of AstroGrid science (calculating Redshift) presented in [1], a reduction of 43.47% (data transfer) was achieved by enacting a workflow using a decentralized architecture. Hence, in the context of scientific applications, we believe a fault tolerant decentralized solution towards workflow execution should be the way forward.

With the advent of cloud computing, concepts such as *Elastic Computing*, *federation of clouds*, and *Mobile Cloud Computing*, are just around the corner. In respect to the scientific workflows, the benefit of decentralization comes in the context of Elastic Computing. Consider, a compute intensive workitem is processing huge volumes of data. If a centralized orchestrator is handling this task, then the orchestrator is responsible for gathering resources, provision them into the existing workflow, perform migrations at remote locations and several others resulting in an un-necessary burden on the controlling authority. Moreover, a centralized engine foster a lot of infrastructure, development, maintainence and operational costs. Therefore, for certain scientific experiments limited by budget constraints, the decentralized architecture is an ideal candidate.

Bell *et al.* [3] states “the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, ‘*workflow management*’, visualization, and cloud computing technologies”. Taking this line of reasoning as a benchmark for the proposed work, we propose a decentralized solution to execute a scientific workflow via membrane computing approach. We take inspiration from biology to design a workflow management model that execute services, realizing the *process-steps* or *workitems*, in a decentralized manner. We use the elementary principles of membrane computing to model the execution of a workflow. The choice of this type of an architecture comes with the benefit that it provides a natural and an intuitive method to model workflows, pass parameters and elastically increase or decrease resources at runtime. We consider each membrane as a service capable of executing a scientific workitem. The membranes are self capable of discov-

¹ <http://www.lsst.org/lst/science>

ering other resources (or services) on their own. Each service is provided with *evolutionary rules* (Membrane terminology), it is through these rules the services evolve, allocate & provision extra resources, and execute their respective tasks in a cost-efficient manner.

Substantial literature focuses on achieving scientific workflow execution over the cloud [1], [2], [9], [11], [12]. However, most of these rely on the cloud (or the orchestrator) to elastically increase or decrease resources at runtime, or don't focus on elasticity at all. Therefore, our objective is to introduce '*autonomy*' in resource provisioning by the work-items themselves.

To the best of our knowledge, this is the first endeavour to explore the possibility of executing a workflow via Membrane Computing. To demonstrate the viability of the same in actual deployment, we have developed a prototype with 'real' services. We execute real scientific workflows collected from myexperiment², and deploy the prototype on a virtualized platform (XENServer) to test the validity of the proposed work. During validation, the services exchange data and parameters via a stable storage. The stable storage itself was offered as a service, thereby affirming to the standards of Service-Oriented Architecture. Moreover, using Software Defined Networking, we study the effects of limited bandwidth capabilities during execution.

The contribution of this paper is two-fold:

- 1) A novel membrane inspired approach for decentralizing workflow execution, with autonomous provisioning of computing resources
- 2) A proof of concept, via actual deployment and execution of scientific workflows.

The rest of the paper is organized as follows: Section ii presents a brief introduction to membrane computing. Membrane Inspired Scientific Workflow execution is discussed in Section iii. Results are discussed in Section iv, Related Work is presented in Section v. We conclude with Future Work in Section vi.

For the purpose of clarification, a scientific workflow is called a workflow throughout this paper. A resource is analogous to a service instantiated on a Virtual Machine.

2 Membrane Computing Paradigm

Before beginning the discussion of Membrane Inspired workflow management, we present a small discussion on the Membrane Computing paradigm.

Membrane computing takes its inspiration from a living cell. A living cell is encapsulated by a membrane that separate its internals from the external environment. The cell encloses multiple natural artifacts, e.g. nucleus, golgi apparatus, molecules, vesicles etc. The cytoplasm holds charged ions, whose movement (either inwards or outwards) is controlled by the presence of certain type of proteins. Using chemical receptors, the membrane allows a selective passage of molecules and achieves cell-to-cell signaling.

² <http://myexperiment.org>

The pioneering work in the area of membrane computing was proposed in [6]. The author proposed, the basic structure of a membrane consist of several separate sub-membranes. The membranes consist of the delimiting region, called multiset, where several different objects are placed. The evolution, manipulation, and transformation of objects is accomplished via evolutionary rules. The objects are transferred from one membrane to another membrane, causing transitions and carrying out the intended tasks. The execution rules are chosen in a non-deterministic manner, thereby presenting an illusion of having infinite parallel computing capability. The application of these rules is conditional i.e. a rule is invoked if certain reaction conditions are applicable. The rules as explained in [6] are of the form $a \rightarrow b$, where a and b represents multisets of objects. Since, the data and objects are transferred from one membrane to another, the author proposed the notion of ‘target indications’. Using target indications, the objects are retained, transferred and consumed. It can be deduced that using these rules the multiset can be written very easily. An example of a rule applied towards object evolution is demonstrated below.

Consider a rule of the form $(ij) \rightarrow (i,here) (i,out) (j,in) (j,out)$. In this example, a copy of i and j is consumed and two copies of i and j are produced. One copy of i is retained within the same membrane (the ‘here’ indicator), while the other one moves out to the surrounding environment (the ‘out’ indicator). Out of the two copies of j produced, one goes to the surrounding environment and the other moves inwards toward the inner membrane(s). There exists catalytic rules demonstrating the applicability only in the presence of a certain type of an object, e.g. $cb \rightarrow cv$, where c is the catalyst. Also, there are non co-operating rules, e.g. $a \rightarrow b$, membrane dissolving rules, e.g. $j \rightarrow o\delta$, where δ denotes the membrane dissolving action. It should noted here, the author [6] deliberately points out that the membrane dissolving rule cannot be applied to the skin membrane (for obvious reasons). Further, there are communication rules, symport and antiport, demonstrating how membranes communicate. As outlined earlier, in the real world the membranes communicate via protein channels. Therefore, the protein channel and the molecules are the agents of communication in membrane computing. The ‘*symport rules*’ allows for the passage of molecules in one way. On the other hand, the ‘*antiport rules*’ allow for a two way communication via molecules.

There also exists membrane division and merging rules. A membrane division rule is of the form $[_1 a]_1 \rightarrow [_2 b]_2 [_3 c]_3$ (a membrane is denoted as ‘[]’, [6]), while a membrane merging rule is of the form $[_2 e]_2 [_3 f]_3 \rightarrow [_1 d]_1$. Further, there exists endocytosis rules, exocytosis rules, gemmation rules etc. The rules are applied locally in each membrane in a non-deterministic, maximally parallel manner, thereby causing *transitions* in the system.

In this paper, we try to use these elementary concepts to achieve a decentralized workflow execution. Based on the discussion so far, it is understood that this paradigm has a natural orientation, and can execute any type of computation problem (e.g. Satisfiability [14], Traveling Salesman Problem [16] etc). It is due to this feature, it has received a lot of attention in literature, right from

the moment of its inception. It can be deduced that the membrane computing paradigm allows a natural metaphor and an intuitive mechanism to model the complex behavior of scientific workflows. As discussed previously the paradigm allows communication rules (symport and antiport), membrane dissolving rules, membrane division and merging rules etc. Using these rules, scientific workflow constructs and functionality can be managed with little efforts. Further, applying the evolutionary rules, a workflow itself can be modified dynamically (via endocytosis, exocytosis, gemmation etc.).

3 Membrane inspired scientific workflow execution

In the proposed work, a membrane is considered as a service capable of realizing a single ‘workitem’ or a ‘*process-step*’. Each membrane has its fluid (local memory) capable of storing the contextual information and local data (or molecules). The contextual information includes the load-indicator parameters, the inner membranes (the successor workitems), the outer membranes (the predecessor workitems), the resource pool etc (discussed below). The membranes communicate via the ‘symport’ rules to pass control to the subsequent membrane. The objects and data are passed via the multiset. The data is equivalent to the proteins capable of penetrating the membrane structure. The membranes do not pass data directly, but rather direct the subsequent membranes to read from the stable storage location (implemented as a scalable distributed shared memory). In the following text, a *service* is called a ‘*membrane*’ for the rest of the paper.

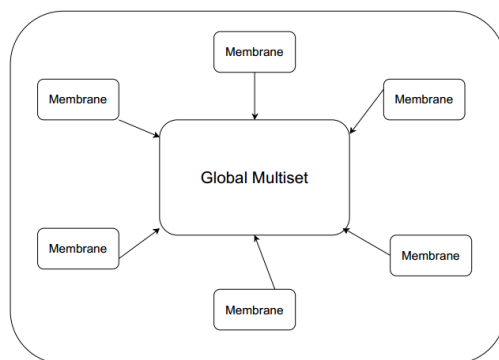


Fig. 1. Membrane Architecture

Before proceeding any further, we must outline the basic architecture utilized to achieve a decentralized execution. Fig 1 represents the overall design used while executing a workflow. The outermost membrane represents the skin membrane, the inner membranes demonstrates the individual workitems. As visible, all membranes pass data and parameters to the global multiset. Each membrane

operates on the objects available in the multiset locally. Hence, there is no need of a central controlling authority. After completing an execution, the membrane dissolves and leaves the transformed objects in the multiset. This procedure is followed till the objects are pushed out to the surrounding i.e. the execution of a workflow has completed. In the membrane computing model, we focus on

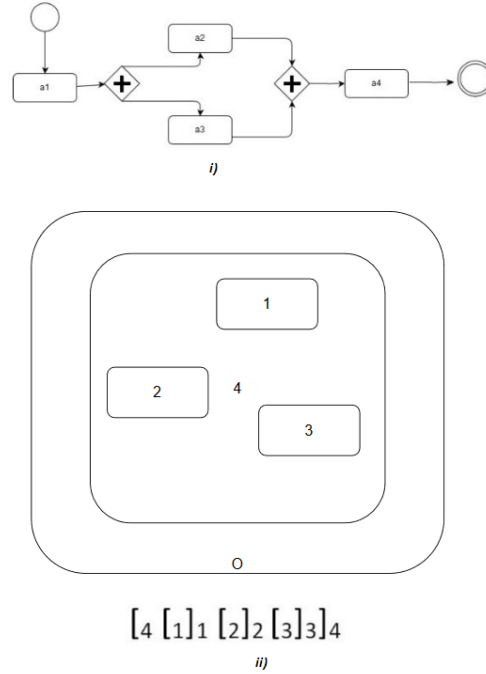


Fig. 2. A Simple Workflow & Membrane Representation

the property of elastically provisioning resources, by the membranes themselves using cell division rules. An example of cell division rules was outlined previously. Using such type of division rules, the membrane is instantly divided, and the load is dynamically shared between the parent membrane and the newly born child membranes. It should be pointed out here that the provisioning of resources is done autonomously (i.e. by a membrane itself), thereby the need of the cloud service provider to locate and provision resources is removed completely. While provisioning extra resources (resource elasticity property), or in membrane terminology, when a parent membrane is dynamically divided into child membranes, the child membranes also read and write to the same multiset. In this way, a membrane is self capable of finding and provisioning resources into existing workflow. Previous works in literature do not take this property into consideration, they either depend on cloud service provider or on the orchestrator to

accomplish this functionality. On the other hand, we focus on self-provisioning of resources by the membranes themselves.

During execution, when a membrane is dynamically divided, the load has to be shared between parent and child membranes. Since the provisioning of resources, or rather, the membrane division process is autonomous (no orchestrator), therefore one might think that the division process will take some time. However, in the results section we prove that the division process, rather than slowing things down, actually speeds up the process.

To dynamically divide a membrane, a *load-indicator* is instantiated with each membrane. When a threshold value is reached, for either the response time, the throughput, the queue size etc., the membrane division rule is invoked and the parent membrane is divided into multiple child membranes. Whenever, the load-indicator sensed the load has reduced, the membrane merging rule was invoked and the extra provisioned resources were released.

In the proposed model, the multiset is considered as a stable, semantically distributed shared memory offered as a service. Semantic space allow an inherent capability for parallel processing and a distributed stable data management platform. This type of platform allows huge volumes of data to be stored in a semantic format, with event-driven and asynchronous mode of communication. Such type of a storage schema allows a simple Application Programming Interface (API) to read and write to a persistent storage. The APIs to access the protocol are exposed as a web service (SOAP & RESTful APIs). The data center allows the query to specified in GData, OpenSearch, XQuery/XPath over HTTP, SPARQL etc. Since, the discussion of semantic spaces is out of scope for this paper, we direct the interested reader to [7], [8].

$$\begin{aligned}
 [1a]_1 &\longrightarrow [2b,in]_2 [3c,in]_3 \delta \\
 [2b]_2 &\longrightarrow [4d,in]_4 \delta \\
 [3c]_3 &\longrightarrow [4e,in]_4 \delta \\
 [4de]_4 &\longrightarrow [af,out]_o \delta
 \end{aligned}$$

Fig. 3. Execution Rules

To demonstrate the procedure used in the membrane computing model to execute a workflow is explained with the help of an example. A simple workflow with four activities in BPMN (Business Process Management Notation) [15] format is demonstrated in Fig 2. The equivalent representation using membranes is shown in the same Figure. The rules to execute the workflow are shown in Fig 3. The rules here do not include any division behavior, rather they specify the execution procedure from a global point of view. It should be pointed out here that the elastic behavior or the cell division rules are specified locally to each membrane.

In this Figure, a, b, c, d, e, f are multisets of objects. The execution begins when membrane one performs a transition, and writes the transformed data to the multiset (Rule 1). Further, using Rule 1 the membrane dissolved (the δ indicator)

and let membranes two and three access the multiset to read the objects b and c respectively. A similar procedure is followed for all the execution rules. The last rule demonstrates the completion of transition activities, and the system halts. In the proposed work, we have used the notion of the *dependency operator*, enabling a sense of determinism during execution. In this paper, we propose the use dependency operator as $\xrightarrow[\text{Dependency}]{\text{Details}}$, where Dependency represents the type of dependency (data or control) and Details demonstrate the list of dependent membranes. It is a known fact that the membrane at same level execute their rules in a non-deterministic manner. However, owing to control dependencies, certain membranes have to be restricted from execution. In such scenarios, the proposed operator provides a certain level of determinism and restrict few of the membranes from execution. For example, consider the following dependency rule:

$$[1[S_1 a]_{S_1}[S_2 b]_{S_2}[S_3 c]_{S_3}]_1 \xrightarrow[\text{Control}]{S_2, S_3} [1[S_1 a]_{S_1}]_1 \delta \quad (1)$$

In this rule, the dependency is control and the membrane list contains $\{S_2, S_3\}$. The rule implies that prior to the execution of S_1 , or, in other words, prior to the application of this rule, both S_2 and S_3 must dissolve (or S_1 and S_2 must have completed their resp. tasks).

It was outlined previously, the membranes are self sufficient to procure resources on their own. To accomplish this functionality, cell division rules are utilized. For example, consider a rule

$$[1 a]_1 \longrightarrow [1.1 a]_{1.1} [1.2 a]_{1.2}$$

In this example, membrane one is divided into two halves, each having the same processing functionality and capability. The division process is carried out using symport rules, involving the membrane and the load-indicator. As soon the membrane divided, the execution procedure and order became

$$[1.1 a]_{1.1} [1.2 a]_{1.2} \longrightarrow [2 b, in]_2 [3 c, in]_3 \delta$$

If, in the middle of a transaction, the provisioned resources have to be released, then cell merging rules are invoked, e.g.

$$[1.1 a]_{1.1} [1.2 a]_{1.2} \longrightarrow [1 a]_1$$

Hence, by applying the division rules and the merging rules, resources can be provisioned and released. The specification and application of these rules is dependent on each individual membrane, there is no authority that controls the division and merging process. The membranes are self-sufficient and self-capable of invoking division rules and merging rules independently. During real world experimentation, we have also used the same procedure to execute the scientific workflows.

In the membrane computing model, a specific role is assigned to each membrane. Moreover, the membranes are assigned a unique name and an identifier. Membranes assigned to the same role can execute the same functionality. In

the *Future Internet*, the issue of reliability is inevitable, therefore redundant membranes should be kept as back-up in case one fails during execution. Next, while executing a workflow there are certain input and output dependencies that must be resolved before proceeding. In the proposed work, these dependencies are specified in an XML format thereby providing a straightforward mapping to a machine readable format. Since, a lot of work [1], [5], [13] has been done to resolve dependencies and automate the execution of traditional workflows, therefore we rely on those procedures to proceed with the execution.

Listing 1.1. ResourcePool

```

<ResourcePool>
  <Resource>
    <Address>
      http://10.200.40.139/Traffic/Diverte/node1
    </Address>
    <Endpoint>
      .
      .
    </Endpoint>
  </Resource>
  <Resource>
    <Address>
      http://10.200.40.132/Traffic/Diverte/node2
    </Address>
    <Endpoint>
      .
      .
    </Endpoint>
  </Resource>
</ResourcePool>

```

Now, to begin with the execution, a workflow is specified to the multiset. Every participating membrane reads its corresponding dependencies (a low level locking mechanism). In the experiments, we used an XML schema. It should be noted that our motive is *'not'* to introduce a new *description language*. The schema is not limited and was constructed using the principles of domain specific languages. Therefore, any type of workflow can be mapped to a machine readable and executable format, thereby presenting a language independent interface. In that case, each membrane must be equipped to handle any type of description. A question arises here: How do membranes understand these specifications? To interpret these constructs, each membrane is equipped with a local interpreter. Hence, an extra layer is added to the membranes to correctly interpret the workflow description (either XML or normal rules).

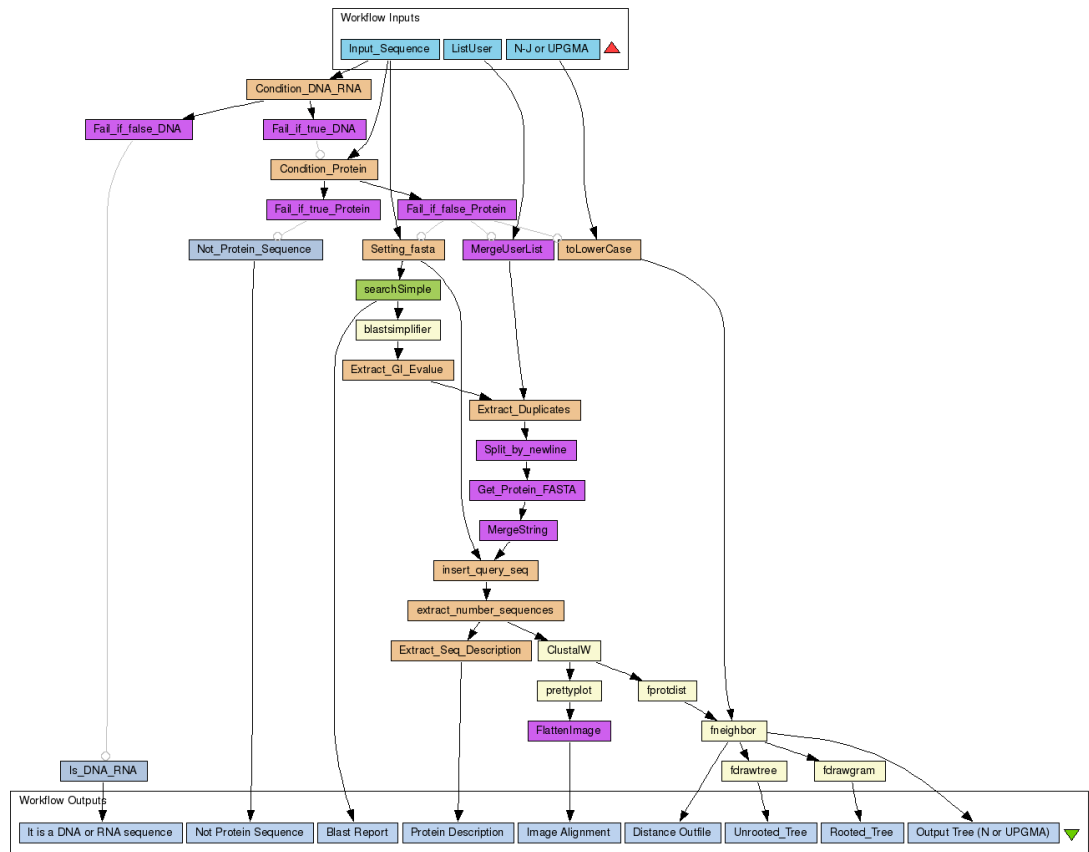


Fig. 4. One of the Workflows for Experimentation

4 Results

4.1 Experimental Setup

In order to evaluate the proposed work, and to check the viability in actual deployment scenarios, we have conducted experiments with multiple workflows collected from myexperiment.org. The execution of these workflows was achieved in 1) Inside the Institute’s Intranet 2) Virtual Machines within the Computing Lab of the Institute. The configuration of each machine in the Intranet is i5 Processor with 4 GB RAM, whereas the resource pool had multiple machines, each having Quad-Core processors with 8GBs and 16GBs of RAM. The distributed shared memory, MozartSpaces³, was deployed as a RESTful service. The application container for the services was Apache Tomcat v7.0.41. The experiments were conducted specifically keeping in mind the capability to provision resources independently, specifically we concentrate on Infrastructure as a Service.

To study the effect of a low bandwidth environment, the networking capabilities of each Virtual Machine (VM) was constrained. In this paper, one of our motive was to test the model’s performance and feasibility to execute a decentralized scientific workflow under duress with limited bandwidth capabilities. We discuss the behavior of the model in the next subsection.

4.2 Execution Efficiency

As outlined previously, we have chosen workflows from myexperiment. The workflows are uploaded by the people in the research community, and spans different fields viz. Bio-Informatics, Protein Sequence Analysis, nucleotide and protein sequence analysis. Each workflow was executed multiple times. A total of 13, 28, and 18 services were developed for workflow I⁴, II⁵, and III⁶.

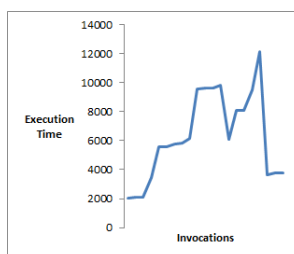


Fig. 5. Execution Time WF-I No Constraints

³ <http://www.mozartspaces.org/>

⁴ <http://www.myexperiment.org/workflows/244.html>

⁵ <http://www.myexperiment.org/workflows/124.html>

⁶ <http://www.myexperiment.org/workflows/1477.html>

During the experiments, it was assumed that each of the discrete workitems are realized by a web service. In Fig 5, we have shown the execution time of the workflow I, with no constraints on the bandwidth capabilities. As visible, the execution time of the workflow started at a normal pace. But, when the invocations increased linearly, the execution time followed. The moment, the load-indicator (*of each individual membrane*) sensed duress, a new resource or a Virtual Machine (VM) was provisioned from the resource pool. The newly provisioned resource was made aware of the stable storage location and access methodology.

After provisioning resources, the execution time experienced a sudden drop. This is clearly visible in Fig 5. It should be noted here that in the experiments, each workitem (or membrane) provisioned resources on its own. The first reduction in the execution time is due to the fact that only a few membranes provisioned an extra resource. Therefore, the drop is not that much steep. However, a sudden decrement in the execution time at the end of the graph indicate multiple VMs were provisioned to complete the workflow. We invoked the same workflow multiple times (in regular intervals) so as to test the behavior of autonomously provisioning resources. It should be noted here, when the membranes provisioned extra resources, it happened when the load indicator sensed duress for the *new* incoming request. The already existing requests were not dynamically *migrated* (live migration). In business terminology, the SLAs (Service Level Agreements) were violated for the new requests only, there is no need to provision resources for non-SLA violating requests (*principles of cost elasticity*).

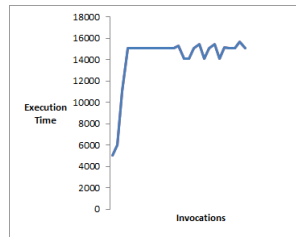


Fig. 6. Execution Time WF-III No Constraints

The same procedure was followed to execute workflows II and III. The result demonstrating the execution time for workflow II is shown in Figure 6. As visible, the execution time dropped the moment the load indicator sensed an increment in the load of the individual membrane. Hence, looking at the execution results for workflows I and II, it can be said that though the membranes provisioned resources autonomously, but the execution efficiency was never compromised. The results demonstrate the feasibility of the membrane computing paradigm towards executing scientific workflows without requiring a centralized controller, or without pre-defined arrangements with the cloud service provider (Infrastructure as a Service).

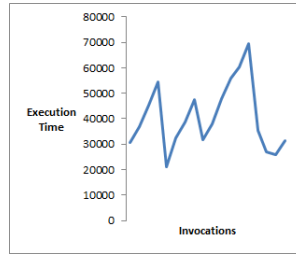


Fig. 7. Execution Time WF-I Limited Bandwidth

In the experiments conducted so far, the bandwidth of each VM was not limited. During experimentation, we limited the bandwidth of each VM deployed to 2KBytes/s. This was done to evaluate the feasibility of the prototype in real world scenarios. The resulting graph for the execution time is shown in Fig 7 (workflow I). As demonstrated in the Figure, the provisioning of extra resources resulted the same sudden drop in the execution time. However, in this case the execution time increased. This effect is due to the fact that the bandwidth is limited and each individual membrane required some time to receive the data dependencies. Moreover, it was observed that there were instances when the request was dropped due to severe congestion.

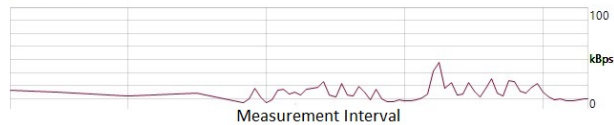


Fig. 8. Network Performance Limited Bandwidth

In Figures 8, we have shown the snapshots of the network performance of two VMs chosen at random. It can be seen from the Figures that the network capabilities witnessed its peak during execution. It is at that instant, the new incoming requests were dropped. Further, the performance showed that whenever the load increased beyond the threshold limit, new VMs were provisioned to balance the load.

5 Related Work

In literature, there are lot of techniques available to execute a workflow, either centrally or decentrally. In the decentralized scenario, the services share data and parameters either by passing messages directly or indirectly. However, only a few are in the context of dynamically provisioning resources for scientific workflows with actual deployment.

Nature inspired metaphors have caught some attention lately. Based on these approaches we found two interesting metaphors 1) Chemistry 2) Physics. A decentralized framework to execute workitems, is proposed in [5], [2]. Fernandez *et al* [2] propose executing a workflow using the chemical paradigm. Similar to our work, the authors used a decentralized environment, however, they used a centralized shared memory (hence, the authors suffered from scalability issues). Moreover, they kept the services fixed to execute the workitems, with no provision of dynamic adaptations. Further, the issues of elasticity is not addressed. A similar method to achieve orchestration and choreography is proposed in [5]. The author used the same chemistry model to achieve orchestration and choreography. Next, the work in physics, focus achieving *motion co-ordination*, using the notion of ‘*Computational Fields*’ [17]. However, the focus is distributed motion co-ordination, not scientific workflow execution. A similar to technique to synchronize the motion of ‘Unmanned aerial system (UAS)’ is proposed in [18]. The author has used the notion of physics inspired co-fields to accomplish this functionality. A similar physics inspired technique is proposed in [10]. The authors also focus achieving a decentralized service composition in dynamic environments. However, the focus is not scientific workflow execution.

A cloud based middleware is proposed in [11]. It is a platform proposed to elastically provision resources at run time. However, the main focus of [11] is not scientific workflow execution in a decentralized environment. In [9], a comparison between the resource consumption between a high performance computing machine and a cloud based environment is presented. The cloud experiments are conducted on Amazon’s EC2. The author found, the resources provisioned from the cloud were not that powerful as compared to a traditional high performance computing machine. They found, though executing scientific workflow was acceptable, however the data transfer costs were high. This is one of the factors we will be focusing in the future works. How to find an optimal balance between resource and cost elasticity? [12] introduces the concept of scheduling data intensive scientific workflows in a cloud based environment with virtual clusters. The scheduling is based on the ‘iterative ordinal optimization’ algorithm. The application of the algorithm produces a significantly lower processing overhead involved in scheduling the workflows. In this paper, we also achieved a decentralized workflow execution based on observable virtual clusters.

6 Conclusion and Future Work

In this paper, we introduced the membrane computing paradigm towards realizing a scientific workflow. The membranes acted independently, with a global perspective, and provisioned resources autonomously. We validated our approach over real services, on real virtualization platform within the computer labs of our institute. We were able to demonstrate that the proposed methodology was effective in provisioning resources autonomously at run-time, thereby validating the technique for an actual deployment. Future work in this direction includes work towards extending the methodology towards incorporating mobile devices.

Further, we are looking towards deploying the services on a grid based facility, while provisioning resources from a public cloud. Finally, we propose to study the effects of cost elasticity on resource provisioning.

7 Acknowledgement

The authors would like to thank fellow Ph. D students, to help with the technical issues faced during experimentation, and figuring out with the technicalities of Membrane Computing. We would also like to thank the people in the research community, for sharing their scientific workflows.

References

1. Barker A., Walton C. D., Robertson D.: "Choreographing web services" *Services Computing, IEEE Transactions on* 2, no. 2 (2009): 152-166.
2. Fernandez H., Tedeschi C., Priol T.: "A Chemistry Inspired Workflow Management System for Decentralizing Workflow Execution", *IEEE Transactions on Services Computing*, doi: 10.1109/TSC.2013.27 (pre-print).
3. Bell G., Hey T., Szalay A.: "Beyond the data deluge." *Science* 323, no. 5919 (2009): 1297-1298.
4. Alonso G., Agrawal D, Abbadi A. E., Mohan C.: "Functionality and limitations of current workflow management systems." *IEEE Expert* 12, no. 5 (1997): 105-111.
5. Wang C., Pazat J.: "A Chemistry-Inspired Middleware for Self-Adaptive Service Orchestration and Choreography." In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 426-433. IEEE, 2013.
6. Paun G.: "Computing with membranes" *Journal of Computer and System Sciences* 61, no. 1 (2000): 108-143.
7. Wang X., Dong J. S., Chin C., Hettiarachchi S. R., Zhang D.: "Semantic space: An infrastructure for smart spaces." *Computing* 1, no. 2 (2002): 67-74.
8. Zhuge H.: "Semantic grid: Scientific issues, infrastructure, and methodology." *Communications of the ACM* 48, no. 4 (2005): 117-119.
9. Juve G., Deelman E., Vahi K., Mehta G., Berriman B., Berman B. P., Maechling P.: "Scientific workflow applications on Amazon EC2." In *E-Science Workshops, 2009 5th IEEE International Conference on*, pp. 59-66. IEEE, 2009.
10. Ahmed T., and Srivastava A. "Minimizing Waiting Time for Service Composition: A Frictional Approach." In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pp. 268-275. IEEE, 2013.
11. Calheiros R. N., Vecchiola C., Karunamoorthy D., Buyya R.: "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds." *Future Generation Computer Systems* 28, no. 6 (2012): 861-870.
12. Zhang F., Cao J., Hwang K., Wu C.: "Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds." In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 9-17. IEEE, 2011.
13. Zaha J. M., Barros A., Dumas M., Hofstede A. T.: "Let's dance: A language for service behavior modeling." In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pp. 145-162. Springer Berlin Heidelberg, 2006.

14. Cecili J. M., Garca J. M., Guerrero G. D., Martnez-del-Amor M., Hurtado I. P., Prez-Jimnez M.: "Simulating a P system based efficient solution to SAT by using GPUs." *The Journal of Logic and Algebraic Programming* 79, no. 6 (2010): 317-325.
15. White S. A.: "Introduction to BPMN." *IBM Cooperation* 2, no. 0 (2004): 0.
16. Nishida T. Y. "An approximate algorithm for NP-complete optimization problems exploiting P systems." In *Proc. Brainstorming Workshop on Uncertainty in Membrane Computing*, pp. 185-192. 2004.
17. Mamei M., Zambonelli F., Leonardi L.: "Distributed motion coordination with co-fields: A case study in urban traffic management." In *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on*, pp. 63-70. IEEE, 2003.
18. Reza H., Ogaard K.: "Modeling UAS swarm system using conceptual and dynamic architectural modeling concepts." In *Conceptual Structures for Discovering Knowledge*, pp. 331-338. Springer Berlin Heidelberg, 2011.

P Systems with Anti-Matter

Artiom Alhazov¹, Bogdan Aman², Rudolf Freund³

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

² Institute of Computer Science, Romanian Academy, Iaşi, Romania
E-mail: bogdan.aman@iit.academiaromana-is.ro

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

Abstract. The concept of a matter object being annihilated when meeting its corresponding anti-matter object is investigated in the context of P systems. Computational completeness can be obtained with using only non-cooperative rules besides these matter/anti-matter annihilation rules if these annihilation rules have priority over the other rules. Without this priority condition, in addition catalytic rules with one single catalyst are needed to get computational completeness. Even deterministic systems are obtained in the accepting case. Allowing anti-matter objects as input and/or output, we even get a computationally complete computing model for computations on integer numbers. Interpreting sequences of symbols taken in from and/or sent out to the environment as strings, we get a model for computations on strings, which can even be interpreted as representations of elements of a group based on a computable finite presentation.

1 Introduction

Membrane systems as introduced in [16] and usually called *P systems* can be considered as distributed multiset rewriting systems, where all objects – if possible – evolve in parallel in the membrane regions and may be communicated through the membranes. Overviews on this emerging field of research can be found in the monograph [17] and the handbook of membrane systems [18]; for actual news and results we refer to the P systems webpage [20]. Computational completeness (computing any partial recursive relation on non-negative integers) can be obtained with using cooperative rules or with catalytic rules (eventually) together with non-cooperative rules. In this paper, we use another concept to avoid cooperative rules in general: for any object a (*matter*), we consider its anti-object (*anti-matter*) a^- as well as the corresponding *annihilation rule* $aa^- \rightarrow \lambda$, which is assumed to exist in all membranes; this annihilation rule could be assumed to remove a pair a, a^- in zero time, but here we use these annihilation rules as special non-cooperative rules having priority over all other rules in the sense of

weak priority (e.g., see [2], i.e., other rules then also may be applied if objects cannot be bound by some annihilation rule any more). The idea of anti-matter has already been considered in another special variant of P systems with motivation coming from modeling neural activities, which are known as spiking neural P systems; for example, spiking neural P systems with anti-matter (*anti-spikes*) were already investigated in [15]. Moreover, in [6] the power of anti-matter for solving NP-complete problems is exhibited.

As expected (for example, compare with the Geffert normal forms, see [19]), the annihilation rules are rather powerful. Yet it is still surprising that using matter/anti-matter annihilation rules as the only non-cooperative rules, with the annihilation rules having priority, we already get computational completeness without using any catalyst; without giving the annihilation rules priority, we need one single catalyst. Even more surprising is the result that with priorities we obtain *deterministic* systems in the case of accepting P systems. Allowing anti-matter objects as input and/or output, we even get a computationally complete computing model for computations on integer numbers. Finally, by interpreting sequences of symbols taken in from and/or sent out to the environment as strings, we also consider P systems with anti-matter as computing/accepting/generating devices for string languages or even for languages over a group based on a computable finite presentation.

2 Prerequisites

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

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

The family of regular and recursively enumerable string languages is denoted by *REG* and *RE*, respectively. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as [5] and [19].

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

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

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

3 P Systems

The basic ingredients of a (cell-like) P system are the membrane structure, the multisets of objects placed in the membrane regions, and the evolution rules. The *membrane structure* is a hierarchical arrangement of membranes, in which the space between a membrane and the immediately inner membranes defines a *region/compartment*. The outermost membrane is called the *skin membrane*, the region outside is the *environment*. Each membrane can be labeled, and the label (from a set *Lab*) will identify both the membrane and its region; the skin membrane is identified by (the label) 1. The membrane structure can be represented by an expression of correctly nested labeled parentheses, and also by a rooted tree (with the label of a membrane in each node and the skin in the root). The *multisets of objects* are placed in the compartments of the membrane structure and usually represented by strings of the form.

The *evolution rules* are multiset rewriting rules of the form $u \rightarrow v$, where $u \in O^\circ$ and $v = (b_1, tar_1) \dots (b_k, tar_k)$ with $b_i \in O^\circ$ and $tar_i \in \{here, out, in\}$ or $tar_i \in \{here, out\} \cup \{in_j \mid j \in Lab\}$, $1 \leq i \leq k$. Using such a rule means “consuming” the objects of u and “producing” the objects from b_1, \dots, b_k of v , where the target *here* means that the objects remain in the same region where the rule is applied, *out* means that they are sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied

in the skin region), *in* means that they are sent to one of the immediately inner membranes, chosen in a non-deterministic way, and in_j means that they are sent into the specified inner membrane. In general, the target indication *here* is omitted.

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

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

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

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

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

In the *generative case*, a halting computation has associated a result, in the form of the number of objects present in membrane l_{out} in the halting configuration (l_{in} can be omitted). In the *accepting case*, for $l_{in} \neq 0$, we accept all (vectors of) non-negative integers whose input, given as the corresponding numbers of objects in membrane l_{in} , leads to a halting computation (l_{out} can be omitted). For the input being taken from the environment, i.e., for $l_{in} = 0$, we need an additional target indication *come*; $(a, come)$ means that the object a is taken into the skin from the environment (all objects there are assumed to be available in an unbounded number). The multiset of all objects taken from the environment during a halting computation then is the multiset accepted by this accepting P system, which in this case we shall call a *P automaton* [4]. The set of non-negative integers and the set of (Parikh) vectors of non-negative integers generated/accepted/accepted in the automaton way as results of halt-

ing computations in Π are denoted by $N_\delta(\Pi)$ and $Ps_\delta(\Pi)$, respectively, with $\delta \in \{gen, acc, aut\}$.

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

Computational completeness for (generating) catalytic P systems can be achieved when using two catalysts or with three catalysts in purely catalytic P systems, and the same number of catalysts is needed for P automata; in accepting P systems, the number of catalysts increases with the number of components in the vectors of natural numbers to be analyzed [8]. It is a long-time open problem how to characterize the families of sets of (vectors of) natural numbers generated by (purely) catalytic P systems with only one (two) catalysts. Using additional control mechanisms as, for example, priorities or promoters/inhibitors, P systems with only one (two) catalyst(s) can be shown to be computationally complete, e.g., see Chapter 4 in [18]. Last year several other variants of control mechanism have been shown to lead to computational completeness in (purely) catalytic P systems using only one (two) catalyst(s), see [7], [10], and [11]. In this paper we are going to investigate the power of using matter/antimatter annihilation rules – with the astonishing result, that no catalysts are needed any more in case the annihilation rules have weak priority over the other rules.

The family of sets $Y_\delta(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, computed by (purely) catalytic P systems with at most m membranes and at most k catalysts is denoted by $Y_\delta OP_m(cat_k)$ ($Y_\delta OP_m(pcat_k)$). The following characterizations are known:

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

$$Ps_{acc}OP_m(cat_{d+2}) = Ps_{acc}OP_m(pcat_{d+3}) = N^d RE.$$

$$Ps_\delta OP_m(cat_2) = Ps_\delta OP_m(pcat_3) = Ps RE.$$

4 Using Matter and Anti-Matter

This concept to be used in (catalytic) P systems is a direct generalization of the idea of anti-spikes from spiking neural P systems (see [15]): for each object a we introduce the anti-matter object a^- . We can look at these anti-matter objects a^- as objects of their own or else we may extend the notion of a (finite) multiset over the (finite) alphabet V , $V = \{a_1, \dots, a_n\}$, as a mapping $f : V \rightarrow \mathbb{N}$ to a mapping $f : V \rightarrow \mathbb{Z}$ now also allowing negative values. In a usual way, such an extended multiset on \mathbb{Z} is represented by $\langle a_1^{f(a_1)}, \dots, a_n^{f(a_n)} \rangle$. A unique string representation for such an extended multiset is obtained by assigning a string over the (ordered) alphabet $\langle a_1, a_1^-, \dots, a_n, a_n^- \rangle$ as $a_1^{f(a_1)} \dots a_n^{f(a_n)}$ such that

$(a_i)^{-m}$, $m > 0$, is represented by $(a_i^-)^m$, $1 \leq i \leq n$. Any other string having the same Parikh vector with respect to the (ordered) alphabet $\langle a_1, a_1^-, \dots, a_n, a_n^- \rangle$ can be used for representing the multiset given by f as well.

As in spiking neural P systems with anti-spikes, also in cell-like P systems we might consider the annihilation of matter and anti-matter objects to happen in zero-time or in an intermediate step between normal derivation steps in the maximally parallel mode. Whenever related matter a and anti-matter a^- meet, they annihilate each other, as, for example, in an extended multiset on \mathbb{Z} matter a and anti-matter a^- cannot exist at the same moment, hence, also not in a string representing an extended multiset on \mathbb{Z} .

Yet in this paper we consider both objects and anti-objects to be handled by usual evolution rules; the annihilation of matter and anti-matter objects then corresponds to an application of the (non-context-free!) rule $aa^- \rightarrow \lambda$. In contrast to the case described above, now in an instantaneous description of a configuration of a P system both matter and anti-matter objects may appear. When working with context-free or catalytic rules over an (ordered) alphabet $\langle a_1, a_1^-, \dots, a_n, a_n^- \rangle$, we may give the matter/anti-matter annihilation rules weak priority over all other rules – in order to not have matter a and anti-matter a^- in some configuration at the same moment and let them “survive” for longer.

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

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

Theorem 2. *For any $n \geq 1$, $k \geq 0$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

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

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

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

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

- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$.

As rules common for all simulations of SUB-instructions, we have

$$a_r^- \rightarrow \#^-, \quad 1 \leq r \leq m,$$

and the annihilation rules

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

as well as the trap rules

$$\#^- \rightarrow \#\# \quad \text{and} \quad \# \rightarrow \#\#;$$

these last two rules lead the system into an infinite computation whenever a trap symbol is left without being annihilated.

The *zero test* for instruction l_1 is simulated by the rules

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

The symbol $\#$ generated by the second rule $l_1' \rightarrow \#l_3$ can only be eliminated if the anti-matter a_r^- generated by the first rule $l_1 \rightarrow l_1' a_r^-$ is not annihilated by a_r , i.e., only if register r is empty.

The *decrement case* for instruction l_1 is simulated by the rule

$$l_1 \rightarrow l_2 a_r^-.$$

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

- $l_h : HALT$. Simulated by $l_h \rightarrow \lambda$.

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

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

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

$$\begin{aligned} Y_\delta OP_n(cat(k), antim) &= YRE \quad \text{and} \\ ZY_\alpha OP_n(cat(k), antim) &= ZYRE. \end{aligned}$$

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

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

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

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

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

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

$$\begin{aligned} l_1 &\rightarrow l_2 a_r^-, \\ l_1 &\rightarrow l_1'' d a_r^-, \\ l_1'' &\rightarrow l_1', \text{ and} \\ l_1' &\rightarrow \# l_3. \end{aligned}$$

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

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

as well as the trap rules

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

but in addition, also

$$d \rightarrow \# \#$$

as well as the catalytic rules

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

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

enforces $a_r a_r^- \rightarrow \lambda$ to be applied, if possible, already in the first step instead of $ca_r^- \rightarrow c\#^-$, as in a successful derivation the catalyst c first has to eliminate the dummy symbol d .

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

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

Theorem 4. *For any $n \geq 1$, $k \geq 0$, and $Y \in \{N, Ps\}$,*

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

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

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

$$\begin{aligned} \tilde{M}_r &= \{\tilde{l} \mid l : (SUB(r), l', l'') \in P\}, & \hat{M}_r &= \{\hat{l} \mid l : (SUB(r), l', l'') \in P\}, \\ \tilde{M}_r^- &= \{\tilde{l}^- \mid l : (SUB(r), l', l'') \in P\}, & \hat{M}_r^- &= \{\hat{l}^- \mid l : (SUB(r), l', l'') \in P\}. \end{aligned}$$

We now take the rules

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

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

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

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

5 When Matter/Anti-Matter Annihilation Generates Energy

The matter/anti-matter annihilation may also be assumed to result in the generation of a specific amount of “energy”, which is also well motivated by physics. In the definitions of these systems, the matter/anti-matter annihilation rules

$a_r a_r^- \rightarrow \lambda$ are replaced by $a_r a_r^- \rightarrow e$ where e is a symbol denoting this special amount of energy.

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

The following results are immediate consequences of the corresponding Theorems 2 and 4 – in both cases, each matter/anti-matter annihilation rule $xx^- \rightarrow \lambda$ is replaced by $xx^- \rightarrow e$ where e is this symbol denoting a special amount of energy, and, in addition, we add the rule $e \rightarrow \lambda$:

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

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

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

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

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

Corollary 3. *The construction in the proof of Theorem 2 can be adapted in such a way that the simulation of each instruction of the register machine takes exactly three steps (including the annihilation rules), and moreover, the number of energy objects e at the end of a successful computation exactly equals the number of instructions simulated.*

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. Following the construction given in the proof of Theorem 2, the instructions of M now can be simulated as follows:

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

$$\begin{aligned} l_1 &\rightarrow l_1', \\ l_1' &\rightarrow l_1'', \\ l_1'' &\rightarrow ea_r l_2, \\ l_1'' &\rightarrow ea_r l_3. \end{aligned}$$

- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$.

As rules common for all simulations of SUB-instructions, we have

$$\begin{aligned} a_r^- &\rightarrow \#^-, 1 \leq r \leq m, \\ a_r a_r^- &\rightarrow e, 1 \leq r \leq m, \\ \#\#^- &\rightarrow e, \\ \#^- &\rightarrow \#\#, \\ \# &\rightarrow \#\#. \end{aligned}$$

The *zero test* for instruction l_1 is simulated by the rules

$$\begin{aligned} l_1 &\rightarrow l_1' a_r^-, \\ l_1' &\rightarrow \#l_1'', \text{ and} \\ l_1'' &\rightarrow l_3; \end{aligned}$$

- the symbol $\#$ generated by the second rule $l_1' \rightarrow \#l_1''$ can only be eliminated if the anti-matter a_r^- generated by the first rule $l_1 \rightarrow l_1' a_r^-$ is not annihilated by a_r , i.e., only if register r is empty; e is generated by $\#\#^- \rightarrow e$.

The *decrement case* for instruction l_1 is simulated by the rules

$$\begin{aligned} l_1 &\rightarrow \tilde{l}_1 a_r^-, \\ \tilde{l}_1 &\rightarrow \tilde{l}_1', \\ \tilde{l}_1' &\rightarrow l_2; \end{aligned}$$

- here, e is generated by $a_r a_r^- \rightarrow e$.
- $l_h : HALT$. Simulated by the rules

$$\begin{aligned} l_h &\rightarrow l_h', \\ l_h' &\rightarrow l_h'', \\ l_h'' &\rightarrow e. \end{aligned}$$

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

Remark 1. Let M be a register machine and

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

Then, according to [3], RS is recursive. Hence, although $L(M)$ may not be recursive, $RS(M)$ is recursive in any case.

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

6 Computing with Integers

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

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

Theorem 5. *For any $d \geq 1$ we have that:*

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

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

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

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

Proof. As we have shown in Section 4, all variants of P systems with anti-matter mentioned in the theorem are computationally complete when dealing with multisets over any arbitrary alphabet, being able to simulate the actions of a register machine. Hence, as any d -dimensional vector of integer numbers can be represented by a $2d$ -dimensional vector of non-negative integers, which can be processed in the usual way by register machines and thus simulated by all the variants of P systems with anti-matter mentioned in the theorem, we

only have to solve the technical detail how to get this $2d$ -dimensional vector of non-negative integers from a given d -dimensional vector of integer numbers represented by symbols over the (ordered) alphabet $\langle a_1, a_1^-, \dots, a_d, a_d^- \rangle$: given the input in an input membrane $\neq 0$, we there just make a first step using in parallel the non-cooperative rules $a_i \rightarrow [a_i, +]$ and $a_i^- \rightarrow [a_i, -]$, $1 \leq i \leq d$. Then the multisets over these symbols can be handled in the usual way, now both of them having the corresponding anti-matter objects $[a_i, +]^-$ and $[a_i, -]^-$. In a similar way, we can take the input from the environment by using rules of the form $q \rightarrow p(a_i, come)[a_i, +]$ or $q \rightarrow p(a_i^-, come)[a_i, -]$ where q, p represent states of the register machine. The symbols a_i and a_i^- then are not needed any more and can be eliminated by the rules $a_i \rightarrow \lambda$ and $a_i^- \rightarrow \lambda$. The remaining computations in the respective P system then can be carried out by simulating the actions of a register machine. \square

7 Computing with Languages

P systems with anti-matter, as most of the computationally complete variants of P systems, can also be considered as language generating devices – the objects sent out can be concatenated to strings over a given alphabet, and the objects taken in during a halting computation can be assumed to form a string. For sake of simplicity, we may assume that in each computation step, at most one symbol is sent out or taken in; otherwise, as usual, e.g., see [4], we may take any permutation of the symbols sent out or taken in to be part of a string to be considered as output or input, respectively. Obviously, according to this method of getting an input string, for the accepting case only the automaton variant is to be considered now, as otherwise we would have to take an encoding of the input string by a multiset.

7.1 Languages over Strings

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

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

Example 1. Consider the P system with anti-matter

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

The reader may easily verify that

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

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

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

Theorem 6. *For any arbitrary alphabet V we have that:*

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

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

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

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

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

7.2 Languages over Computable Finite Presentations of Groups

Strings may be used in a wider sense as representations of group elements. In order to establish these more general results, we first need some definitions and examples from group theory, e.g., see [12].

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

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

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

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

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

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

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

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

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

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

Remark 2. Unfortunately, given a finite presentation of a group $\langle B \mid R \rangle$, in general it is not even decidable whether the group presented in that way is finite or infinite. Hence, in this paper we restrict ourselves to infinite groups where the word equivalence problem $u = v$ is decidable, or equivalently, there is a decision procedure telling us whether, given two strings u and v , $u \circ v^{-1} = e$. In that case, we call $\langle B \mid R \rangle$ a *recursive* or *computable* finite group presentation.

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

Example 6. Consider the P system

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

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

Remark 3. The reader may easily verify that, given any finitely generated Abelian group, such a regular P system exists which generates all strings representing the (unique, with respect to a complete order on the positive generators) irreducible representations of the group elements. For non-commutative groups with relators, such trivial representations are not possible.

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

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

$$\begin{aligned} \Pi &= (\{q_0\}, []_1, q_0, R_1, 1) \text{ where} \\ R_1 &= \{q_0 \rightarrow \lambda\} \cup \{q_0 \rightarrow gq_0 \mid g \in B\}. \end{aligned}$$

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

Remark 4. In general, as long as we have given the group by a computable finite presentation, for a mechanism having the full power of Turing computability, we can require that the “strings” sent out to the environment are irreducible ones. Hence, for a given recursively enumerable set L of elements over the computable finite presentation $\langle B \mid R \rangle$ of a group, such a mechanism can generate the irreducible string representations of the elements in L . Thus, the results collected in the following theorem are obvious consequences of the results stated in Theorem 6.

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

Theorem 7. *Let $\langle B \mid R \rangle$ be the computable finite presentation of a group. Then we have that:*

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

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

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

$$\begin{aligned} L_\delta^{\langle B \mid R \rangle}OP_n(cat(k), antim) &= RE^{\langle B \mid R \rangle} \text{ and} \\ ZL_{aut}^{\langle B \mid R \rangle}OP_n(cat(k), antim) &= ZRE^{\langle B \mid R \rangle}. \end{aligned}$$

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

Remark 5. Let us mention that the results obtained in Theorem 7 for arbitrary computable finite presentations $\langle B \mid R \rangle$ of a group can also be applied to the infinite Abelian groups \mathbb{Z}^d with their canonical group presentations by the unit vectors $(0, \dots, 1, \dots, 0)$ and their inverses $(0, \dots, -1, \dots, 0)$. Keeping in mind that there is a one-to-one correspondence between the representation of a vector in \mathbb{Z}^n by a multiset of symbols and the corresponding string representing this multiset, most of the results shown in Theorem 5 are special cases of the respective results stated in Theorem 7.

8 Summary

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

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

Acknowledgements. The authors gratefully acknowledge the inspiring ideas and discussions with Gheorghe Păun on the topics exhibited in this paper; even more results on P systems with anti-matter can be found in [1].

References

1. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and Anti-Matter in Membrane Systems. In: L. F. Macías-Ramos, M. A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, 2014, 1–26.

2. A. Alhazov, D. Sburlan: Static Sorting P Systems. In: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.): Applications of Membrane Computing. *Natural Computing Series*, Springer, 2005, pp. 215–252.
3. M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-Related Outputs of Computations in P Systems. *Journal of Automata, Languages and Combinatorics* **11** (3), 263–278 (2006).
4. E. Csuhaj-Varjú, Gy. Vaszil: P Automata or Purely Communicating Accepting P Systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing. International Workshop, WMC-CdeA 2002 Curtea de Argeş, Romania, August 19–23, 2002. Revised Papers*. Lecture Notes in Computer Science **2597**, Springer, 2003, pp. 219–233.
5. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
6. D. Díaz-Pernil, F. Peña-Cantillana, M. A. Gutiérrez-Naranjo: Antimatter as a Frontier of Tractability in Membrane Computing. *Brainstorming Week in Membrane Computing*, Sevilla, February 2014.
7. R. Freund: Purely Catalytic P Systems: Two Catalysts Can Be Sufficient for Computational Completeness. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin (Eds.): *CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013*. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 2013, pp. 153–166.
8. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science* **330**, 251–266 (2005).
9. R. Freund, M. Oswald: A Small Universal Antiport P System with Forbidden Context. In: H. Leung, G. Pighizzini (Eds.): *8th International Workshop on Descriptive Complexity of Formal Systems - DCFS 2006*, Las Cruces, New Mexico, USA, June 21 - 23, 2006. Proceedings DCFS, New Mexico State University, Las Cruces, New Mexico, USA, 2006, pp. 259–266.
10. R. Freund, M. Oswald: Catalytic and Purely Catalytic P Automata: Control Mechanisms for Obtaining Computational Completeness. In: S. Bensch, F. Drewes, R. Freund, F. Otto (Eds.): *Fifth Workshop on Non-Classical Models of Automata and Applications (NCMA 2013)*, OCG, Wien, 2013, pp. 133–150.
11. R. Freund, Gh. Păun: How to Obtain Computational Completeness in P Systems with One Catalyst. In: T. Neary and M. Cook: *Proceedings Machines, Computations and Universality 2013, MCU 2013*, Zürich, Switzerland, September 9–11, 2013, *EPTCS* **128**, 47–61 (2013).
12. D. F. Holt, B. Eick, E. A. O'Brien: *Handbook of Computational Group Theory*. CRC Press, 2005.
13. I. Korec: Small Universal Register Machines. *Theoretical Computer Science* **168**, 267–301 (1996).
14. M. L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
15. L. Pan, Gh. Păun: Spiking Neural P Systems with Anti-Matter. *International Journal of Computers, Communications & Control* **4** (3), 273–282 (2009).
16. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, www.tucs.fi).
17. Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
18. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

19. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*, 3 volumes. Springer, 1997.
20. The P Systems Website: www.ppage.psystems.eu.

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

Artiom Alhazov¹ and Rudolf Freund²

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

² Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
E-mail: rudi@emcc.at

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

1 Introduction

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

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

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

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

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

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

2 Definitions

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

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

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

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

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

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

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

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

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

Note 1. The definition above is taken from [1]. As it will be shown in Remark 2, without restricting generality, every set P_j may be assumed to be a singleton. The meaning of a **set** of promoters may be defined differently, replacing “there exists some $p \in P_j$ ” by “for all $p \in P_j$ ” in the definition above. This alternative definition corresponds to definitions of sets as permitting context in string rewriting models, yet, such a promoter set would be equivalent to a singleton promoter which is a union of this set, so (unless we stress the descriptorial complexity) also in this case we do not need to deal with multi-element promoters. In the rest of the paper we assume the first interpretation, as in the definition above, noting that this variation does **not** influence results from [1] or those in this paper.

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

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

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

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

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

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

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

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

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

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

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

even in the the case of a deterministic P system: If the first promoter is chosen to make the rule r applicable, we do not care about the other promoters; if the second promoter is chosen to make the rule r applicable, we do not allow p_1 to appear in the configuration, but do not care about the other promoters p_3 to p_m ; in

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

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

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

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

3 Results

In this section we first recall several results from [1] and then we establish our new results, first for deterministic P systems with non-cooperative rules and only priorities as control mechanism, and then we characterize systems using promoters or inhibitors of weight 2.

3.1 Recent Results

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Deterministic Non-Cooperative P Systems with Priorities Only

We start with an example showing how an object t can be rewritten in a deterministic way depending on the presence or absence of an object a .

Example 1.

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

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

We now proceed with characterizing systems with priorities only.

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

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

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

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

We proceed with the converse inclusion. Let $\Pi_0 = (\{a, t\}, \{a\}, t, R, >)$, then $R = \{t \rightarrow t\}$ and the empty relation $>$ yield \emptyset . To accept $\{0\}$, it is enough to instead set $R = \{a \rightarrow a\}$ and again take the empty relation $>$.

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

Using the idea from Example 1, we now construct the system

$$\begin{aligned} \Pi_1 &= (O, \Sigma = \{a_{i,0} \mid 1 \leq i \leq k\}, tA_{0,0} \cdots A_{k,0}, R, >), \text{ where} \\ O &= \{a_{i,j} \mid 1 \leq i \leq k, 0 \leq j \leq i+1\} \cup \{A_{i,j} \mid 0 \leq i \leq k, 0 \leq j \leq i+2\} \\ &\quad \cup \{t, z, p\} \cup \{t_i \mid 0 \leq i \leq k+1\}, \\ R &= R_a \cup R_A \cup R_t, \\ R_a &= \{a_{i,j} \rightarrow a_{i,j+1} \mid 1 \leq i \leq k, 0 \leq j \leq i\} \\ R_A &= \{A_{i,j} \rightarrow A_{i,j+1} \mid 1 \leq i \leq k, 0 \leq j \leq i+1\} \\ R_t &= \{t \rightarrow t_0, t_0 \rightarrow z, t_0 \rightarrow t_1, p \rightarrow p\} \cup \{t_i \rightarrow t_{i+1}, t_i \rightarrow p \mid 1 \leq i \leq k\}, \\ > &= \{a_{i,0} \rightarrow a_{i,1} > A_{0,0} \rightarrow A_{0,1} \mid 1 \leq i \leq k\} \\ &\quad \cup \{A_{0,0} \rightarrow A_{0,1} > t_0 \rightarrow z, A_{0,1} \rightarrow A_{0,2} > t_0 \rightarrow t_1\} \\ &\quad \cup \{a_{i,i} \rightarrow a_{i,i+1} > A_{i,i} \rightarrow A_{i,i+1} \mid 1 \leq i \leq k\} \\ &\quad \cup \{A_{i,i} \rightarrow A_{i,i+1} > t_i \rightarrow p, A_{i,i+1} \rightarrow A_{i,i+2} > t_i \rightarrow t_{i+1} \mid 1 \leq i \leq k\}. \end{aligned}$$

Such a system accepts exactly $\mathbb{N}_k \cup \{0\}$. Indeed, after the first step, $A_{0,1}$ is present if all input symbols were absent, otherwise $A_{0,0}$ is still present instead. For any i , $1 \leq i \leq k$, after step $1 + i$, object $A_{i,i}$ is present if input symbol $a_{i,0}$ was present in the input, and otherwise $A_{i,i+1}$ is present instead. These “decision symbols” are used by t_i , $0 \leq i \leq k$, to build the “presence picture”. We recall that it suffices to accept when all input symbols are present, or when none is present. In the second case, t_0 becomes z , and the computation only continues by rules from R_A , leading to halting. Now let us assume that the first s of the input symbols are present, $s < k$. Then, t_0 becomes $t_1 \cdots t_s$, yet at the end the absence of t_{s+1} causes t_s to change into p , leading to an infinite computation. But if all input symbols are present, then finally the computation halts with t_{k+1} .

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

3.3 Deterministic Non-Cooperative P Systems with Promoters or Inhibitors of Weight 2

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

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

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

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

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

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

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

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

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

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

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

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

4 Conclusion

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

Acknowledgements

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

References

1. A. Alhazov, R. Freund: Asynchronous and Maximally Parallel Deterministic Controlled Non-Cooperative P Systems Characterize *NFIN* and *coNFIN*. *The Tenth Brainstorming Week in Membrane Computing*, vol. 1, Sevilla, 2012, 25–34, and *Membrane Computing - 13th International Conference*, CMC13, Budapest (E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil, Eds.), *Lecture Notes in Computer Science* **7762**, 2013, 101–111.
2. A. Alhazov, D. Sburlan: Ultimately Confluent Rewriting Systems. Parallel Multiset-Rewriting with Permitting or Forbidding Contexts. In: G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa: *Membrane Computing, 5th International Workshop*, WMC 2004, Milano, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, 2005, 178–189.
3. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient, *Theoretical Computer Science* **330**, 2, 2005, 251–266.
4. R. Freund, M. Kogler, M. Oswald, A General Framework for Regulated Rewriting Based on the Applicability of Rules. In: J. Kelemen, A. Kelemenová, *Computation, Cooperation, and Life*, Springer, *Lecture Notes in Computer Science* **6610**, 2011, 35–53.
5. R. Freund, S. Verlan: A Formal Framework for Static (Tissue) P Systems. *Membrane Computing, 8th International Workshop*, WMC 2007 Thessaloniki, 2007, Revised Selected and Invited Papers (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science* **4860**, 2007, 271–284.
6. O.H. Ibarra, H.-C. Yen: Deterministic Catalytic Systems are Not Universal, *Theoretical Computer Science* **363**, 2006, 149–161.
7. M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
8. Gh. Păun: *Membrane Computing. An Introduction*, Springer, 2002.
9. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
10. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
11. P systems webpage. <http://ppage.psystems.eu>

P Systems with Toxic Objects

Artiom ALHAZOV¹ and Rudolf FREUND²

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

² Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
E-mail: rudi@emcc.at

Abstract. We introduce the new concept of toxic objects, objects that must not stay idle as otherwise the computation is abandoned without yielding a result. P systems of many kinds using toxic objects allow for smaller descriptonal complexity, especially for smaller number of rules, as trap rules can be avoided. Besides presenting a number of tiny P systems generating or accepting non-semilinear sets of (vectors of) natural numbers with very small numbers of rules, we also improve the results for catalytic and purely catalytic P systems: 14 rules for generating a non-semilinear vector set and 29 rules for generating a non-semilinear number set are sufficient when allowing only the minimal number of two (three) catalysts; moreover, with using toxic objects, these numbers can be reduced to 11 and 17. Yet only 23 rules – without using toxic objects – are needed if we allow more catalysts, i.e., five (seven) catalysts.

1 Introduction

P systems using non-cooperative rules without any additional control have the behaviour of *EOL* systems, but when only taking the results at halting means that the objects evolve in a context-free manner, generating *PsCF*, which is known (by Parikh's theorem) to coincide with *PsREG*, i.e., to the family of semilinear sets. In the accepting setup, P systems using non-cooperative rules without any additional control are even weaker, accepting all multisets over some subalphabet, or nothing, see [1].

In [2] we were interested in P systems able to generate or accept non-semilinear sets of natural numbers or at least sets of vectors of natural numbers, yet with as small ingredients as possible for different variants of P systems. Our main focus was on the descriptonal complexity of these P systems, i.e., on how small the total number of rules may be, depending on the specific features of particular models of P systems. As most of the models of P systems can be shown to be computationally complete based on a simulation of the actions of a register machine, even simple examples often turn out to be somehow more complicated than the general proof.

As an example of special interest we consider the case of (purely) catalytic P systems: Ideas how to find good examples of catalytic P systems generating a non-semilinear set of numbers were discussed intensively during the Fourteenth International Conference on Membrane Computing (CMC 2013) in Chişinău, especially by Petr Sosík and Rudolf Freund, based on a draft by Petr Sosík, and most of them finally being included in his paper [21]. Some new observations just found recently allowed us to reduce the number of rules again in a considerable way, see Section 4.6. Using the concept of toxic objects allows for saving a lot more rules. As a second interesting example we consider P systems with target selection and give another example which shows how toxic objects may help to save a lot of rules.

In this paper, we investigate the new concept of *toxic objects* (first mentioned in [2]) which allows us to “kill” a computation branch if we cannot find a multiset of rules covering all occurrences of toxic objects which then somehow become “lethal” by killing such a computation. For all the proof techniques using a trap symbol $\#$ to “kill” a computation by introducing the trap symbol $\#$ with a non-cooperative rule $a \rightarrow \#$, the concept of toxic objects allows us to save most of the trap rules, thus improving the descriptive complexity of the underlying P systems.

The rest of the paper is organized as follows: We first recall the basic definitions from formal language theory as well as the definitions for the most important variants of P systems considered later. Then we present examples with a small number of rules for variants of P systems generating or accepting a non-semilinear set of natural numbers or of vectors of natural numbers with only a few rules, especially for the maximally parallel derivation mode, explained in more detail in [2]. As a specific example for a variant of P systems where the use of toxic objects allows for saving a lot of rules we consider P systems with label selection. Then, for the catalytic and purely catalytic P systems, we improve previous results established in [21] and [22].

2 Definitions

In this section we first recall the basic notions from formal language theory needed in this paper and then the definitions of the basic variants of P systems considered in the following sections. For more details in formal language theory we refer the reader to the standard monographs and textbooks as [19] and for the area of regulated rewriting to [8]. All the main definitions and results for P systems can be found in [16] and [17]; only specific notations and models not yet to be found there will be explained in more detail in this paper, especially the new idea of “*toxic objects*”, which will be explained and studied in Section 4. For actual informations and new developments in the area of membrane computing we refer to the P systems webpage [23].

2.1 Prerequisites

The set of non-negative integers (*natural numbers*) is denoted by \mathbb{N} . An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V , the free monoid generated by V under the operation of concatenation is denoted by V^* ; the elements of V^* are called strings, and the *empty string* is denoted by λ ; $V^* \setminus \{\lambda\}$ is denoted by V^+ . Let $\{a_1, \dots, a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in a string x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to a_1, \dots, a_n is $(|x|_{a_1}, \dots, |x|_{a_n})$. The *Parikh image* of a language L over $\{a_1, \dots, a_n\}$ is the set of all Parikh vectors of strings in L , and we denote it by $Ps(L)$. For a family of languages FL , the family of Parikh images of languages in FL is denoted by $PsFL$; for families of languages over a one-letter alphabet, the corresponding sets of non-negative integers are denoted by NFL ; for an alphabet V containing exactly d objects, the corresponding sets of Parikh vectors with d components is denoted by N^dFL , i.e., we replace Ps by N^d .

A (finite) *multiset* over the (finite) alphabet V , $V = \{a_1, \dots, a_n\}$, is a mapping $f : V \rightarrow \mathbb{N}$ and represented by $\langle f(a_1), a_1 \rangle \dots \langle f(a_n), a_n \rangle$ or by any string x the Parikh vector of which with respect to a_1, \dots, a_n is $(f(a_1), \dots, f(a_n))$. In the following we will not distinguish between a vector (m_1, \dots, m_n) , its representation by a multiset $\langle m_1, a_1 \rangle \dots \langle m_n, a_n \rangle$ or its representation by a string x having the Parikh vector $(|x|_{a_1}, \dots, |x|_{a_n}) = (m_1, \dots, m_n)$. Fixing the sequence of symbols a_1, \dots, a_n in the alphabet V in advance, the representation of the multiset $\langle m_1, a_1 \rangle \dots \langle m_n, a_n \rangle$ by the string $a_1^{m_1} \dots a_n^{m_n}$ is unique. The family of regular, context-free, and recursively enumerable string languages is denoted by REG , CF , and RE , respectively.

ET0L systems An *ET0L system* is a construct $G = (V, T, w, P_1, \dots, P_m)$ where $m \geq 1$, V is an alphabet, $T \subseteq V$ is the terminal alphabet, $w \in V^*$ is the *axiom*, and the P_i , $1 \leq i \leq m$, are finite sets (*tables*) of non-cooperative rules over V . In a derivation step in G , all the symbols present in the current sentential form are rewritten using one table. The language generated by G , denoted by $L(G)$, consists of all terminal strings $w \in T^*$ which can be generated by a derivation in G starting from the axiom w . The family of languages generated by *ET0L systems* and by *ET0L systems* with at most k tables is denoted by $ET0L$ and ET_k0L , respectively.

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

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

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

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

Register machines provide a simple universal computational model, for example, see [14]. In the following, we shall call a specific model of P systems *computationally complete* or *universal* if and only if for any register machine M we can effectively construct an equivalent P system Π of that type simulating M and yielding the same results.

Non-semilinear sets of numbers and of vectors of numbers In most of the examples described in the following sections, we will use (variants) of the set of natural numbers

$$\{2^n \mid n \geq 0\} = N \left(\left\{ a^{2^n} \mid n \geq 0 \right\} \right)$$

and the set of (two-dimensional) vectors of natural numbers

$$\{(n, m) \mid n \geq 1, n \leq m \leq 2^n\} = Ps(\{(a^n b^m) \mid n \geq 1, n \leq m \leq 2^n\}),$$

which both are known to not be semilinear.

2.2 P Systems

The ingredients of the basic variants of (cell-like) P systems are the membrane structure, the objects placed in the membrane regions, and the evolution rules. The *membrane structure* is a hierarchical arrangement of membranes. Each membrane defines a *region/compartment*, the space between the membrane and the immediately inner membranes; the outermost membrane is called the *skin membrane*, the region outside is the *environment*, also indicated by (the label) 0. Each membrane can be labeled, and the label (from a set Lab) will identify both the membrane and its region. The membrane structure can be represented by a rooted tree (with the label of a membrane in each node and the skin in the root), but also by an expression of correctly nested labeled parentheses. The *objects* (multisets) are placed in the compartments of the membrane structure and usually represented by strings, with the multiplicity of a symbol corresponding to the number of occurrences of that symbol in the string. The basic *evolution rules* are multiset rewriting rules of the form $u \rightarrow v$, where u is a multiset of objects from a given set O and $v = (b_1, tar_1) \dots (b_k, tar_k)$ with $b_i \in O$ and $tar_i \in \{here, out, in\}$ or $tar_i \in \{here, out\} \cup \{in_j \mid j \in Lab\}$, $1 \leq i \leq k$. Using

such a rule means “consuming” the objects of u and “producing” the objects b_1, \dots, b_k of v ; the *target indications here, out,* and *in* mean that an object with the target *here* remains in the same region where the rule is applied, an object with the target *out* is sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied in the skin region), while an object with the target *in* is sent to one of the immediately inner membranes, non-deterministically chosen, whereas with in_j this inner membrane can be specified directly. In general, we may omit the target indication *here*.

Yet there are a lot of other variants of rules we shall consider later; for example, if on the right-hand side of a rule we add the symbol δ , the surrounding membrane is dissolved whenever at least one such rule is applied, at the same moment all objects inside this membrane (the objects of this membrane region together with the whole inner membrane structure) are released to the surrounding membrane, and the rules assigned to the dissolved membrane region get lost. Another option is to add *promoters* $p_1, \dots, p_m \in O^+$ and *inhibitors* $q_1, \dots, q_n \in O^+$ to a rule and write $u \rightarrow v|_{p_1, \dots, p_m, \neg q_1, \dots, \neg q_n}$, which rule then is only applicable if the current contents of the membrane region includes any of the promoter multisets, but none of the inhibitor multisets; in most cases promoters and inhibitors are rather taken to be singleton objects than multisets. Further variants of P systems will be defined later.

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

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, f)$$

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

If a rule $u \rightarrow v$ has at least two objects in u , then it is called *cooperative*, otherwise it is called *non-cooperative*. In *catalytic P systems* we use non-cooperative as well as *catalytic rules* which are of the form $ca \rightarrow cv$, where c is a special object which never evolves and never passes through a membrane (both these restrictions can be relaxed), but it just assists object a to evolve to the multiset v . In a *purely catalytic P system* we only allow catalytic rules. For a catalytic as well as for a purely catalytic P system Π , in the description of Π we replace “ O ” by “ O, C ” in order to specify those objects from O which are the catalysts in the set C . As already explained above, cooperative and non-cooperative as well as catalytic rules can be extended by adding promoters and/or inhibitors, thus yielding rules of the form $u \rightarrow v|_{p_1, \dots, p_m, \neg q_1, \dots, \neg q_n}$.

All the rules defined so far can be used in different derivation modes: in the *sequential* mode (*sequ*), we apply exactly one rule in every derivation step; in the *asynchronous* mode (*asyn*), an arbitrary number of rules is applied in parallel; in the *maximally parallel* (*maxpar*) derivation mode, in any computation step of Π we choose a multiset of rules from the sets R_1, \dots, R_m in a non-deterministic

way such that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the membrane regions $1, \dots, m$.

The membranes and the objects present in the compartments of a system at a given time form a *configuration*; starting from a given *initial configuration* and using the rules as explained above, we get *transitions* among configurations; a sequence of transitions forms a *computation* (we often also say *derivation*). A computation is *halting* if and only if it reaches a configuration where no rule can be applied any more. With a halting computation we associate a *result generated* by this computation, in the form of the number of objects present in membrane f in the halting configuration. The set of multisets obtained as results of halting computations in Π working in the derivation mode $\delta \in \{sequ, asyn, maxpar\}$ is denoted by $mL_{gen,\delta}(\Pi)$, the set of natural numbers obtained by just counting the number of objects in the multisets of $mL_{gen,\delta}(\Pi)$ by $N_{gen,\delta}(\Pi)$, and the set of (Parikh) vectors obtained from the multisets in $mL_{gen,\delta}(\Pi)$ by $Ps_{gen,\delta}(\Pi)$.

Yet we may also start with some additional input multiset w_{input} over an *input alphabet* Σ in membrane f , i.e., in total we there have $w_f w_{input}$ in the initial configuration, and *accept* this input w_{input} if and only if there exists a halting computation with this input; the set of multisets accepted by halting computations in

$$\Pi = (O, \Sigma, \mu, w_1, \dots, w_m, R_1, \dots, R_m, f)$$

working in the derivation mode δ is denoted by $mL_{acc,\delta}(\Pi)$, the corresponding sets of natural numbers and of (Parikh) vectors are denoted by $N_{acc,\delta}(\Pi)$ and $Ps_{acc,\delta}(\Pi)$, respectively.

The family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\gamma \in \{gen, acc\}$ computed by P systems with at most m membranes working in the derivation mode δ and with rules of type X is denoted by $Y_{\gamma,\delta}OP_m(X)$.

For example, it is well known (for example, see [15]) that for any $m \geq 1$, for the types of non-cooperative (*ncoo*) and cooperative (*coo*) rules we have

$$NREG = N_{gen,maxpar}OP_m(ncoo) \subset N_{gen,maxpar}OP_m(coo) = NRE.$$

For $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$, the family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by (purely) catalytic P systems with at most m membranes and at most k catalysts is denoted by $Y_{\gamma,\delta}OP_m(cat_k)$ and $Y_{\gamma,\delta}OP_m(pcat_k)$, respectively; from [10] we know that, with the results being sent to the environment (which means taking $f = 0$), we have

$$Y_{gen,maxpar}OP_1(cat_2) = Y_{gen,maxpar}OP_1(pcat_3) = YRE.$$

If we allow a catalyst c to switch between two different states c and \bar{c} we call c a bi-stable catalyst; in that way we obtain *P systems with bi-stable catalysts*. For $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$, the family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by (purely) catalytic P systems with bistable catalysts with at most m membranes and at most k catalysts is denoted by $Y_{\gamma,\delta}OP_m(2cat_k)$

and $Y_{\gamma,\delta}OP_m(p2cat_k)$, respectively. We note that in the generative case we do not count the catalysts in the output membrane region.

For all the variants of P systems of type X , we may consider to label all the rules in the sets R_1, \dots, R_m in a one-to-one manner by labels from a set H and to take a set W containing subsets of H . Then a *P system with label selection* is a construct

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, H, W, f)$$

where $\Pi' = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, f)$ is a P system as defined above, H is a set of labels for the rules in the sets R_1, \dots, R_m , and $W \subseteq 2^H$. In any transition step in Π we first select a set of labels $U \in W$ and then apply a non-empty multiset R of rules such that all the labels of these rules in R are in U (and in the case of maximal parallelism, the set R cannot be extended by any further rule with a label from U so that the obtained multiset of rules would still be applicable to the existing objects in the membrane regions $1, \dots, m$). The families of sets $Y(\Pi)$, $Y \in \{N, Ps\}$, computed by P systems with label selection with at most m membranes and rules of type X as well as $card(W) \leq k$ are denoted by $Y_{\gamma,\delta}OP_m(X, ls_k)$, for any $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$.

For all variants of P systems using rules of some type X , we may consider systems containing only rules of the form $u \rightarrow v$ where $u \in O$ and $v = (b_1, tar) \dots (b_k, tar)$ with $b_i \in O$ and $tar \in \{here, out, in\}$ or $tar \in \{here, out\} \cup \{in_j \mid j \in H\}$, $1 \leq i \leq k$, i.e., in each rule there is only one target for all objects b_i ; if catalytic rules are considered, then we request the rules to be of the form $ca \rightarrow c(b_1, tar) \dots (b_k, tar)$; in both cases, for a rule $u \rightarrow (b_1, tar) \dots (b_k, tar)$ we then write $u \rightarrow (b_1, \dots, b_k; tar)$.

A *P system with target selection* contains only these forms of rules; moreover, in each computation step, for each membrane region i we choose a multiset of rules from R_i having the same target indication tar ; for different membrane regions these targets may be different; moreover, the total multiset obtained in this way must not be empty. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by P systems with target selection with at most m membranes and rules of type X are denoted by $Y_{\gamma,\delta}OP_m(X, ts)$, for any $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$.

Remark 1. P systems with target selection were first defined in [11], but there the chosen multiset of rules for each R_i had to be non-empty if possible. In this paper, we only require the total multiset of rules, obtained by choosing multisets of rules in each R_i with the results going to a chosen target membrane, to be non-empty. Yet as in [11] we assume that when choosing the target *in* all objects are sent to just one selected inner membrane.

We may extend rules of the form $u \rightarrow (b_1, \dots, b_k; tar)$ to rules of the form $u \rightarrow (b_1, \dots, b_k; Tar)$ where Tar is a finite set of targets, thus obtaining *P systems with target agreement*. In each computation step, for each membrane we first choose a target tar and then a multiset of rules of the form $u \rightarrow (b_1, \dots, b_k; Tar)$

with $tar \in Tar$ – again, for different membranes these targets may be different. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by P systems with target agreement with at most m membranes and rules of type X are denoted by $Y_{\gamma,\delta}OP_m(X, ta)$, for any $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$.

P systems with target agreement have the same computational power as P systems with target selection, as proved in the following theorem, yet they allow for a more compact description of rules as we will see in Subsection 3.2.

Theorem 1. *For all types of rules X , any $\gamma \in \{gen, acc\}$, any derivation mode $\delta \in \{sequ, asyn, maxpar\}$, any $Y \in \{N, Ps\}$, and any $m \in \mathbb{N}$, we have*

$$Y_{\gamma,\delta}OP_m(X, ta) = Y_{\gamma,\delta}OP_m(X, ts).$$

Proof. Given a P system with *target selection*

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, f)$$

we can also interpret Π as a P system

$$\Pi' = (O, \mu, w_1, \dots, w_m, R'_1, \dots, R'_m, f)$$

with *target agreement* by replacing each rule $u \rightarrow (b_1, \dots, b_k; tar)$ in any of the sets R_i , $1 \leq i \leq m$, by the corresponding rule $u \rightarrow (b_1, \dots, b_k; \{tar\})$ in R'_i . Obviously, $Y_{\gamma,\delta}(\Pi) = Y_{\gamma,\delta}(\Pi')$.

On the other hand, given a P system

$$\Pi' = (O, \mu, w_1, \dots, w_m, R'_1, \dots, R'_m, f)$$

with *target agreement* we immediately get the corresponding P system with *target selection*

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, f)$$

such that $Y_{\gamma,\delta}(\Pi) = Y_{\gamma,\delta}(\Pi')$: for each rule $u \rightarrow (b_1, \dots, b_k; Tar) \in R'_i$ we take all the rules $u \rightarrow (b_1, \dots, b_k; tar)$ with $tar \in Tar$ into R_i . \square \square

Whereas for most of the other variants considered in this paper the so-called *flattening* procedure (for more details see [11]) allows for finding equivalent systems with only one membrane, for P systems with target selection or target agreement the membrane structure usually plays an essential rôle.

For any of the families of (vectors of) natural numbers $Y_{\gamma,\delta}OP_m(X)$ we will add subscript k at the end to indicate that only systems with at most k rules are considered, i.e., we write $Y_{\gamma,\delta}OP_m(X)_k$. If any of the finite parameters like m and k is unbounded, we replace it by $*$.

3 Examples for P Systems with a Small Number of Rules

In this section, we give examples for P systems with a very small number of rules accepting or generating a non-semilinear set (of vectors) of natural numbers. In [2], most of the examples comprised in the following table were elaborated in detail; for the models not defined above and the examples for these models, we refer the reader to this paper.

n	models
1	– deterministic cooperative rules, accepting numbers with a special (non-standard) halting and accepting condition – non-cooperative rules with target agreement, generating numbers
2	– non-cooperative rules with target selection, generating numbers – non-cooperative rules with label selection, generating numbers – non-cooperative rules with tables, generating numbers – non-cooperative rules with membrane dissolution, generating numbers – non-cooperative rules in active membranes with two polarizations, generating numbers – non-cooperative rules with one inhibitor, generating numbers
3	– non-deterministic cooperative rules, accepting numbers – symport/antiport rules of weight ≤ 2 (and size ≤ 4), accepting numbers – non-cooperative rules with one promoter, generating numbers
4	– purely catalytic rules with one bi-stable catalyst, generating vectors – symport/antiport rules of weight ≤ 2 (and size ≤ 3), accepting numbers
5	– non-cooperative rules with promoters/inhibitors, generating numbers
9	– [purely] catalytic rules with one bi-stable catalyst and toxic objects, generating numbers
11	– (purely) catalytic rules with two (three) catalysts and toxic objects, generating vectors
12	– [purely] catalytic rules with one bi-stable catalyst, generating numbers
14	– (purely) catalytic rules with two (three) catalysts, generating vectors
17	– (purely) catalytic rules with two (three) catalysts and toxic objects, generating numbers
23	– (purely) catalytic rules with five (seven) catalysts, generating numbers
29	– (purely) catalytic rules with two (three), generating numbers

Table 1. Examples for using n rules in specific models of P systems.

3.1 Accepting P Systems

Consider the P system

$$\begin{aligned} \Pi_0 &= (O = \{a, s\}, \Sigma = \{a\}, \mu = []_1, w_1 = \lambda, R_1, f = 1) \text{ where} \\ R_1 &= \{aa \rightarrow a, a \rightarrow s, ss \rightarrow ss\}. \end{aligned}$$

In each derivation step, Π_0 halves (part of) the objects a , renaming the rest of objects a into s . If more than one s is produced, an infinite computation is

forced due to the rule $ss \rightarrow ss$. The only way to produce not more than one s is to always have even multiplicity of objects a until it reaches the last one; hence, we conclude $mL_{acc,maxpar}(\Pi_0) = \{a^{2^n} \mid n \geq 0\} \cup \{\lambda\}$.

This accepting P system Π_0 has 3 cooperative rules, i.e., we have

$$\{2^n \mid n \geq 0\} \cup \{0\} \in N_{acc,maxpar}OP_1(coo)_3.$$

It is well-known that P systems with cooperative rules are computationally complete. The smallest known universal one has 23 rules, see [6].

3.2 Generating by Doubling in the Maximally Parallel Mode

This section contains models of P systems with “mass influence”, where something can simultaneously affect (directly or indirectly) an unbounded number of copies of specific objects (e.g., target agreement, target selection, label selection, tables, membrane dissolution). In that way we obtain tiny P system using massive parallelism for repeated doubling, and using one of the effects mentioned above to halt.

P Systems with Target Agreement Or Target Selection We first consider the case of *target agreement* which allows for the smallest descriptonal complexity with only one rule:

$$\begin{aligned} \Pi_2 = (O = \{a\}, \mu = []_1, w_1 = a, w_2 = \lambda, R_1, R_2 = \emptyset, f = 0) \text{ with} \\ R_1 = \{a \rightarrow (aa, \{here, out\})\}. \end{aligned}$$

Π_2 doubles the number of objects each turn that the objects stay in the skin membrane choosing the target *here*. At some moment, all objects agree in the target destination *out* thus moving into the environment where no rule can be applied any more. At any moment of the computation, all simultaneously produced objects must agree in the same destination, effectively choosing between continuing the doubling or halting.

If we resolve the rule $a \rightarrow (aa, \{here, out\})$ into its two corresponding rules $a \rightarrow (aa, here)$ and $a \rightarrow (aa, out)$, we immediately get the P system with *target selection*

$$\begin{aligned} \Pi_3 = (O = \{a\}, \mu = []_1, w_1 = a, w_2 = \lambda, R_1, R_2 = \emptyset, f = 0) \text{ with} \\ R_1 = \{a \rightarrow (aa, here), a \rightarrow (aa, out)\}. \end{aligned}$$

In sum, we therefore infer

$$\{2^n \mid n \geq 1\} \in N_{gen,maxpar}OP_1(ncoo, ta)_1 \cap N_{gen,maxpar}OP_1(ncoo, ts)_2.$$

P Systems with Tables The following P system with tables of rules (*tabled P system*) needs only 2 rules and is closely related to the P system with target selection described in Subsection 3.2.

$$\begin{aligned} \Pi_4 = (O = \{a\}, \mu = []_1, w_1 = b, R_1, f = 0) \text{ with} \\ R_1 = \{T_1 = \{a \rightarrow aa, here\}, T_2 = \{a \rightarrow a, out\}\}. \end{aligned}$$

Π_4 doubles the multiplicity of objects a each step as long as using table T_1 . At any time, if after $n \geq 0$ such steps the second table T_2 is chosen, all objects a are sent out to the environment and the computation halts, having generated a^{2^n} .

We also note that the second table (and thus the table feature) is not needed under a specific variant of halting called *unconditional halting* which resembles the L systems (Lindenmayer systems) mode of taking the result; P systems using non-cooperative rules and taking the results after each computation step (i.e., with unconditional halting) were considered in [7] and shown to characterize $PsETOL$.

In sum, we have obtained

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_1(ncoo, table_2)_2 \cap N_{gen-u,maxpar}OP_1(ncoo)_1,$$

where $gen - u$ indicates that we take the results generated in the output membrane after every computation step (i.e., with unconditional halting) and $table_2$ indicates that we are using 2 tables.

P systems with Label Selection Instead of selecting different targets, in the P system with label selection

$$\Pi_5 = (O = \{a\}, \mu = []_1, w_1 = a, R_1, H = \{1, 2\}, W = \{\{1\}, \{2\}\}, f = 0) \text{ with } R_1 = \{1 : a \rightarrow (aa, here), 2 : a \rightarrow (aa, out)\}.$$

we select different labels to be able to choose when to send out all objects from the skin membrane to the environment; hence, we have

$$\{2^n \mid n \geq 1\} \in N_{gen,maxpar}OP_1(ncoo, ls_2)_2.$$

We will return to this model of P systems with label selection later when considering catalytic rules only.

3.3 P Systems with Membrane Dissolution

When using only non-cooperative rules we cannot obtain computational completeness with the additional feature of membrane dissolution; yet, in [9] an infinite hierarchy with respect to the number of membranes was established using membrane dissolution in a linear membrane structure. Now consider

$$\Pi_6 = (O = \{a\}, \mu = [[]_2]_1, w_1 = \lambda, w_2 = a, R_1 = \emptyset, R_2, f = 1)$$

where $R_2 = \{a \rightarrow aa, a \rightarrow aa\delta\}$. Π_6 doubles the number of objects each turn when only using the rule $a \rightarrow aa$ until at some moment the inner membrane may be dissolved by at least for one a using the dissolution rule $a \rightarrow aa\delta$, thus stopping the computation. This generative system has 2 non-cooperative rules and membrane dissolution and computes $mL_{gen,maxpar}(\Pi_6) = \{a^{2^n} \mid n \geq 0\}$, i.e.,

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_2(ncoo, \delta)_2.$$

4 P Systems with Catalysts

P systems with catalysts were already considered in the originating papers for membrane systems, see [15]. In [10], two (three) catalysts were shown to be sufficient for getting computational completeness with (purely) catalytic P systems. Whether or not one (two) catalyst(s) might already be enough to obtain computational completeness, is still one of the most challenging open problems in the area of P systems. We only know that purely catalytic P systems (working in the maximally parallel mode) with only one catalyst simply correspond with sequential P systems with only one membrane, hence, to multiset rewriting systems with context-free rules, and therefore can only generate linear sets.

Using additional control mechanisms as, for example, priorities or promoters/inhibitors, P systems with only one catalyst can be shown to be computationally complete, e.g., see Chapter 4 of [17]. On the other hand, additional features for the catalyst may be taken into account; for example, we may use bi-stable catalysts (catalysts switching between two different states) as will be considered next.

4.1 Generating a Non-Semilinear Number Set with One Bi-Stable Catalyst

We now turn our attention to the – in general – more difficult generative case, using the maximally parallel mode.

$$\begin{aligned} \Pi_7 &= (O, C = \{c, \bar{c}\}, \mu = []_1, w_1 = \bar{c}pa, R_1, f = 1) \text{ where} \\ O &= \{c, \bar{c}\} \cup \{a, b\} \cup \{p, q, s, t\} \cup \{\#\}, \\ R_1 &= \{ca \rightarrow \bar{c}bb, s \rightarrow t, \bar{c}t \rightarrow cs, cs \rightarrow \bar{c}p, t \rightarrow \#, \# \rightarrow \#, \\ &\quad \bar{c}b \rightarrow ca, p \rightarrow q, cq \rightarrow \bar{c}p, \bar{c}p \rightarrow cs, \bar{c}p \rightarrow \bar{c}, q \rightarrow \#\}. \end{aligned}$$

Π_7 works consists in two phases. In phase 1, in addition to the bi-stable catalyst toggling between c and \bar{c} , there is a state object present, toggling between s and t . Every two steps, a is replaced by bb with c changing to \bar{c} while s changes to t ; then $\bar{c}t$ are reset to cs . If objects a are no longer present, c is idle for one step, and then cs change to $\bar{c}p$, entering phase 2.

In phase 2, the state object toggles between p and q . Every two steps, b is renamed into a with \bar{c} changing to c while p changes to q ; then cq are reset to $\bar{c}p$. If objects b are no longer present, \bar{c} is idle for one step, and then either $\bar{c}p$ change to cs , returning to phase 1, or \bar{c} erases p thus causing the system to halt.

In both phases, if the bi-stable catalyst “chooses” a “wrong” rule, then either t or q are left to themselves, forcing the system to enter an infinite computation, so this computation does not lead to a result any more; hence, in sum we obtain $mL_{gen,maxpar}(\Pi_7) = \{a^{2^n} \mid n \geq 0\}$.

Π_7 is a purely catalytic P system with only one bi-stable catalyst and 12 rules, hence, we have

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_1(2cat_1)_{12}.$$

The computational completeness of P systems with one bi-stable catalyst working in the maximally parallel mode was established in [3].

4.2 P Systems with Toxic Objects – a Special Variant of Halting for Avoiding the Trap Symbol

Throughout this section, a main part of the constructions (which is inevitable for proving computational completeness, too) is the introduction of the trap symbol $\#$ in case the derivation goes the wrong way and by the rule $\# \rightarrow \#$ (or $c\# \rightarrow c\#$ with a catalyst c) guaranteeing the derivation never to halt. Yet most of these rules can be avoided if we apply a specific *new halting strategy* allowing the system to halt without yielding a result. This somehow corresponds to the case of a Turing machine halting its computation in a non-final state, and as for Turing machines, where each such computation halting in a non-final state can be transformed into an infinite computation, in P systems usually the trap rules perform this task to yield an infinite computation.

This specific *new halting strategy* allowing the system to halt without yielding a result can be defined in various ways, e.g., with a special symbol (like the trap symbol) appearing in the configuration. As our main goal now is to save as many rules as possible, we want to stop one step earlier, i.e., before the trap symbol would be introduced. Hence, we specify a specific subset of *toxic* objects O_{tox} ; the P system is only allowed to continue a computation from a configuration C by using an applicable multiset of rules covering all copies of objects from O_{tox} occurring in C ; moreover, if there exists no multiset of applicable rules covering all toxic objects, the whole computation having yielded the configuration C is abandoned, i.e., no results can be obtained from this computation.

This idea somehow resembles the strategy of using priorities with having the rules involving the objects from O_{tox} having priority over other rules. Yet this strategy is both weaker and stronger compared with the strategy of priority on rules: on one hand, we can only specify priorities with respect to the objects from O_{tox} ; on the other hand, if not all copies of *toxic* objects from O_{tox} can be covered by any multiset of rules, we stop without getting a result, whereas in the case of priorities any multiset of rules respecting the priority relation can be used to continue the computation.

For any variant of P systems, we add the set of *toxic* objects O_{tox} and in the specification of the families of sets of (vectors of) numbers generated/accepted by P systems with toxic objects using rules of type X we add the subscript *tox* to O , thus obtaining the families $Y_{\gamma, maxpar} O_{tox} P_m(X)$, for any $\gamma \in \{gen, acc\}$ and $m \geq 1$.

Hence, for the P system with one bi-stable catalyst Π_7 elaborated in the preceding subsection, we obtain the corresponding P system Π_8 with toxic objects

$$\begin{aligned} \Pi_8 &= (O, C = \{c, \bar{c}\}, O_{tox}, \mu = []_1, w_1 = \bar{c}pa, R_1, f = 1) \text{ where} \\ O &= \{c, \bar{c}\} \cup \{a, b\} \cup \{p, q, s, t\}, \\ O_{tox} &= \{q, t\}, \\ R_1 &= \{ca \rightarrow \bar{c}bb, s \rightarrow t, \bar{c}t \rightarrow cs, cs \rightarrow \bar{c}p, \\ &\quad \bar{c}b \rightarrow ca, p \rightarrow q, cq \rightarrow \bar{c}p, \bar{c}p \rightarrow cs, \bar{c}p \rightarrow \bar{c}\}. \end{aligned}$$

According to the arguments established in the preceding subsection, we immediately infer $mL_{gen, maxpar}(\Pi_8) = \{a^{2^n} \mid n \geq 0\}$. This system now does not

need more than 9 rules, and therefore we have

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}O_{tox}P_1(2cat_1)_9.$$

4.3 General Results for [Purely] Catalytic P Systems with Toxic Objects

Looking closer into the computational completeness proofs for catalytic P systems given in [10], we see that the only non-cooperative rules used in the proofs given there are rules involving the trap symbol. When going to purely catalytic P systems, we realize that all rules involving the trap symbol can be assigned to one additional catalyst; for example, to generate any recursively enumerable set of natural numbers we need two catalysts for catalytic P systems and three catalysts for purely catalytic P systems.

As the proof of the basic result

$$PsRE = Ps_{gen,maxpar}OP_1(cat_2) = Ps_{gen,maxpar}OP_1(pcat_3)$$

also is the basis of the construction of the [purely] catalytic P system elaborated in Subsection 4.6, we first recall the main ideas of the simulations for ADD- and SUB-instructions of a register machine $M = (d, B, l_0, h, R)$.

For every instruction label $l \in B$ of the register machine to be simulated two symbols are used to keep the two main catalysts c_1, c_2 busy, starting with $p_j\tilde{p}_j$. At the end, for the halting label h we use the two rules $c_1p_h \rightarrow c_1$ and $c_2\tilde{p}_h \rightarrow c_2$ to stop the derivation in case the simulation has succeeded to be correct. The number n_a stored in register a is represented by n_a copies of the symbol o_a .

Each ADD-instruction $j : (ADD(a), k, l)$, for $a \in \{1, 2, \dots, d\}$, can easily be simulated by the rules $c_1p_j \rightarrow c_1o_ap_k\tilde{p}_k$, $c_1p_j \rightarrow c_1o_ap_l\tilde{p}_l$, and $c_2\tilde{p}_j \rightarrow c_2$.

Each SUB-instruction $j : (SUB(a), k, l)$, only necessary to be considered for $a \in 1, 2$, is simulated in four steps as shown in the table listed below:

Simulation of the SUB-instruction $j : (SUB(a), k, l)$ if register a is not empty	register a is empty
$c_ap_j \rightarrow c_a\tilde{p}_j\tilde{p}'_j$	$c_ap_j \rightarrow c_a\tilde{p}_j\tilde{p}'_j\tilde{p}''_j$
$c_{3-a}\tilde{p}_j \rightarrow c_{3-a}$	$c_{3-a}\tilde{p}_j \rightarrow c_{3-a}$
$c_a o_a \rightarrow c_a c'_a$	$c_a\tilde{p}_j \rightarrow c_a$
$c_{3-a}\tilde{p}_j \rightarrow c_{3-a}$	$c_{3-a}\tilde{p}''_j \rightarrow c_{3-a}\tilde{p}'_j$
$c_a c'_a \rightarrow c_a c''_a$	$c_{3-a}\tilde{p}'_j \rightarrow c_{3-a}\tilde{p}''_j$
$c_{3-a}\tilde{p}'_j \rightarrow c_{3-a}\tilde{p}''_j$	$c_{3-a}\tilde{p}''_j \rightarrow c_{3-a}\tilde{p}'_j$
$c_a\tilde{p}''_j \rightarrow c_a p_k\tilde{p}_k$	$c_a\tilde{p}'_j \rightarrow c_a p_l\tilde{p}_l$
$c_{3-a}c''_a \rightarrow c_{3-a}$	$c_{3-a}\tilde{p}'_j \rightarrow c_{3-a}$

In addition, trap rules guarantee that in case the guess whether the contents of register a is empty or not was wrong, the derivation enters an infinite loop with the rule $\# \rightarrow \#$ in the catalytic case or $c_3\# \rightarrow c_3\#$ in the purely catalytic case. These objects x for which we have such trap rules $x \rightarrow \#$ in the catalytic case or

$c_3x \rightarrow c_3\#$ in the purely catalytic case, are c'_1, c''_1, c'_2, c''_2 and, for every label j of a SUB-instruction $j : (SUB(a), k, l)$, the objects $p_j, p'_j, p''_j, \hat{p}_j, \hat{p}''_j, \bar{p}_j, \bar{p}_j, \bar{p}''_j$, and for every label j of an ADD-instruction $j : (ADD(a), k, l)$ the objects p_j, \tilde{p}_j as well. We should like to mention that these trap rules for the objects p_j, \tilde{p}_j coming from the ADD-instructions $j : (ADD(a), k, l)$ were forgotten to be included in the proof of the special Corollary 8 in [10], whereas in the more general Theorem 4, due to writing down the range of the trap rules in a different way, these trap rules for the symbols indicating an ADD-instruction were included correctly.

The construction shown above strictly follows the proof elaborated in [10]; yet we observe that many symbols are just needed to keep one of the two catalysts busy for one step with being erased or else having to evolve to the trap symbol. Hence, every symbol x for which we only have a rule $c_i x \rightarrow c_i \lambda$, $i \in \{1, 2\}$, can be replaced by just one symbol d_i . In addition, for p_j we now always use c_1 and for \tilde{p}_j (which in fact now is replaced by d_2) we now always use c_2 . Finally, we can also replace all symbols \bar{p}''_j by just one variable \hat{d}_{3-a} .

In that way, we obtain the following tables of rules for the simulation of ADD-instructions and SUB-instructions:

Simulation of the ADD-instruction $j : (ADD(a), k, l)$
 $c_1 p_j \rightarrow c_1 o_a p_k d_2$, $c_1 p_j \rightarrow c_1 o_a p_k d_2$; $c_2 d_2 \rightarrow c_2$.

The table for a SUB-instruction now contains several identical entries:

Simulation of the SUB-instruction $j : (SUB(a), k, l)$ if register a is not empty	register a is empty
$c_1 p_j \rightarrow c_1 d_{3-a} \hat{p}'_j$	$c_1 p_j \rightarrow c_1 d_a \hat{d}_{3-a} \bar{p}''_j$
$c_2 d_2 \rightarrow c_2$	$c_2 d_2 \rightarrow c_2$
$c_a o_a \rightarrow c_a c'_a$	$c_a d_a \rightarrow c_a$
$c_{3-a} d_{3-a} \rightarrow c_{3-a}$	$c_{3-a} \bar{p}''_j \rightarrow c_{3-a} p''_j$
$c_a c'_a \rightarrow c_a d_{3-a}$	$c_{3-a} p''_j \rightarrow c_{3-a} p'_j$
$c_{3-a} \hat{p}'_j \rightarrow c_{3-a} \hat{p}''_j$	$c_a p'_j \rightarrow c_a p_l d_2$
$c_a \hat{p}''_j \rightarrow c_a p_k d_2$	$c_{3-a} \hat{d}_{3-a} \rightarrow c_{3-a}$
$c_{3-a} d_{3-a} \rightarrow c_{3-a}$	

The zero-test case now can be reduced considerably to two steps:

$c_1 p_j \rightarrow c_1 \bar{p}_j$
$c_2 d_2 \rightarrow c_2$
c_a remains idle
$c_{3-a} \bar{p}_j \rightarrow c_{3-a} p_l d_2$

For x being an element of the following set, we have to add the trap rules $c_3 x \rightarrow c_3 \#$ in the purely catalytic case and the corresponding rules $x \rightarrow \#$ in the catalytic case:

$$\{\#, d_1, d_2, c'_1, c'_2\} \cup \{p_j \mid j : (ADD(a), k, l)\} \cup \{p_j, \hat{p}''_j, \bar{p}_j \mid j : (SUB(a), k, l)\}$$

In the case of catalytic P systems, the only non-cooperative rules are these trap rules, and in the case of purely catalytic P systems, the trap rules, and only

those, are associated with the third catalyst c_3 . If we take exactly those objects for which such a trap rule exists as *toxic objects* and omit all trap rules, then we immediately infer the following computational completeness result, where O_{tox} indicates that we are using toxic objects:

Theorem 2. $PsRE = PsO_{tox}P_{gen,maxpar}(cat_2) = PsO_{tox}P_{gen,maxpar}(pcat_2)$.

In general, for all the results elaborated in [10] we obtain similar results: when using toxic objects, then the construction for obtaining the results for catalytic and purely catalytic P systems coincide, as for example in the preceding theorem, where in both cases we only need two catalysts (which number currently is assumed to be the minimal one). Moreover, the simulation becomes somehow deterministic, as only the correct simulation paths survive; in that sense, deterministic register machines can be simulated by *deterministic* [purely] catalytic P systems with toxic objects.

4.4 Generating a Non-Semilinear Vector Set with a [Purely] Catalytic P System

We now construct [purely] catalytic P systems generating the non-semilinear set of pairs of natural numbers $\{(n, m) \mid n \leq m \leq 2^n\}$. First we define the purely catalytic P system Π_9 generating $\{a_3^n a_4^m \mid n \leq m \leq 2^n\}$.

$$\begin{aligned} \Pi_9 = (O, C = \{c_1, c_2, c_3\}, \mu = [\]_1, w_1 = c_1 c_2 c_3 a_1 p_1 d_1 d_2, R_1, f = 1) \text{ where} \\ O = \{c_1, c_2, c_3\} \cup \{a_1, a_2, a_3, a_4, d_1, d_2, p_1, p_2, \#\}, \\ R_1 = \{c_1 a_1 \rightarrow c_1, c_1 p_2 \rightarrow c_1 a_1 a_4 p_2, c_1 p_2 \rightarrow c_1 a_1 a_3 a_4 p_1, c_1 p_2 \rightarrow c_1 a_3 a_4, \\ c_2 a_2 \rightarrow c_2, c_2 p_1 \rightarrow c_2 a_2 a_2 p_1, c_2 p_1 \rightarrow c_2 a_2 a_2 p_2, \\ c_1 d_2 \rightarrow c_1, c_2 d_1 \rightarrow c_2, c_1 d_1 \rightarrow c_1 \#, c_2 d_2 \rightarrow c_2 \#, \\ c_3 p_1 \rightarrow c_3 \#, c_3 p_2 \rightarrow c_3 \#, c_3 \# \rightarrow c_3 \#\}. \end{aligned}$$

This purely catalytic system has 14 rules, and with just omitting the third catalyst we obtain the corresponding catalytic P system having 14 rules, too. But with using toxic objects we can save some of (but in this case not all!) the trap rules, i.e., we obtain the purely catalytic P system with toxic objects Π_{10} with only 11 rules:

$$\begin{aligned} \Pi_{10} = (O, C = \{c_1, c_2\}, O_{tox}, \mu = [\]_1, w_1 = c_1 c_2 a_1 p_1 d_1 d_2, R_1, f = 1) \text{ where} \\ O = \{c_1, c_2\} \cup \{a_1, a_2, a_3, a_4, d_1, d_2, p_1, p_2, \#\}, \\ O_{tox} = \{p_1, p_2, \#\}, \\ R_1 = \{c_1 a_1 \rightarrow c_1, c_1 p_2 \rightarrow c_1 a_1 a_4 p_2, c_1 p_2 \rightarrow c_1 a_1 a_3 a_4 p_1, c_1 p_2 \rightarrow c_1 a_3 a_4, \\ c_2 a_2 \rightarrow c_2, c_2 p_1 \rightarrow c_2 a_2 a_2 p_1, c_2 p_1 \rightarrow c_2 a_2 a_2 p_2, \\ c_1 d_2 \rightarrow c_1, c_2 d_1 \rightarrow c_2, c_1 d_1 \rightarrow c_1 \#, c_2 d_2 \rightarrow c_2 \#\}. \end{aligned}$$

In all cases, the derivations work as follows: the objects p_1, p_2 work as states, and the objects $d_i, i \in \{1, 2\}$, are used to check that the corresponding catalyst

c_i is busy at some stage, otherwise these objects d_i force the system to enter an infinite loop with the trap rules or to *kill* the derivation.

In state p_1 , we decrement (the number of objects representing) register 1 by the rule $c_1a_1 \rightarrow c_1$ and double their number by applying the rule $c_2p_1 \rightarrow c_2a_2a_2p_1$ in parallel. By using the rule $c_2p_1 \rightarrow c_2a_2a_2p_2$ instead, we change to state p_2 , yet without checking whether register 1 is already empty. In case the latter rule is used too late, i.e., if no object a_1 is present any more, then we have to use the trap rule $c_1d_1 \rightarrow c_1\#$.

In state p_2 , we decrement (the number of objects representing) register 2 by the rule $c_2a_2 \rightarrow c_2$ and copy (the contents of) this register to register 1, at the same time adding this number to register 4 by using the rule $c_1p_2 \rightarrow c_1a_1a_4p_2$ in parallel. If this rule is used until no object a_2 is present any more, then we have to use the trap rule $c_2d_2 \rightarrow c_2\#$. If instead we use the rule $c_1p_2 \rightarrow c_1a_1a_3a_4p_1$, we switch back to state 1, at the same moment incrementing register 3, yet again without checking register a_2 for being zero. By using the rule $c_1p_2 \rightarrow c_1a_3a_4$, we end this cycling between states p_1 and p_2 , i.e., now both p_1 and p_2 are not present any more, and the two objects d_1 and d_2 have to be eliminated, which can be achieved by using the two rules $c_1d_2 \rightarrow c_1$ and $c_2d_1 \rightarrow c_2$ in parallel.

At the end of a computation, even if the two objects d_1 and d_2 have already been deleted, the catalysts c_1 and c_2 will still be active to delete all the remaining objects a_1 and a_2 , hence, at the end only copies of objects a_3 and a_4 are present any more. In sum, we conclude that $Ps(\{a_3^n a_4^m \mid n \leq m \leq 2^n\})$ belongs to

$$Ps_{gen,maxpar}OP_1(pcat_3)_{14} \cap Ps_{gen,maxpar}O_{tox}P_1(pcat_2)_{11}.$$

4.5 Purely Catalytic P systems with Label Selection

If we only allow purely catalytic rules with two catalysts, P systems with label selection can apply at most two rules in each derivation step and thus show a behavior rather similar to the way matrix grammars in binary normal form work, i.e., the set of labels defines the matrix of the two rules to be applied together.

We now construct a purely catalytic P system with label selection with two catalysts which generates the non-semilinear set of natural numbers $\{2^n \mid n \geq 1\}$.

$$\begin{aligned} \Pi_{11} &= (O, C = \{c_1, c_2\}, \mu = []_1, w_1 = c_1c_2pa_1, R_1, H, W, f = 1) \text{ where} \\ O &= \{c_1, c_2\} \cup \{a_1, a_2, a_3, p, q, r, \#\}, \\ H &= \{i \mid 1 \leq i \leq 7\} \cup \{i' \mid i \in \{1, 3, 6\}\} \cup \{\#_{a_1}, \#_{a_2}, \#_p, \#_q, \#_r, \#\# \}, \\ W &= \{\{1, 1', \#_q, \#_r\}, \{2, \#_{a_1}, \#_q, \#_r\}, \{3, 3', \#_p, \#_r\}, \{4, \#_{a_2}, \#_p, \#_r\}, \\ &\quad \{5, \#_{a_1}, \#_q, \#_r\}, \{6, 6', \#_p, \#_q\}, \{7, \#_{a_2}, \#_p, \#_q, \#\#\}\}, \\ R_1 &= \{1 : c_2p \rightarrow c_2p, 1' : c_1a_1 \rightarrow c_1a_2a_2, 2 : c_2p \rightarrow c_2q, \\ &\quad 3 : c_2q \rightarrow c_2q, 3' : c_1a_2 \rightarrow c_1a_1, 4 : c_2q \rightarrow c_2p, \\ &\quad 5 : c_2p \rightarrow c_2r, 6 : c_2r \rightarrow c_2r, 6' : c_1a_2 \rightarrow c_1a_3, \\ &\quad 7 : c_2r \rightarrow c_2, \#_{a_1} : c_1a_1 \rightarrow c_1\#, \#_{a_2} : c_1a_2 \rightarrow c_1\#, \#\# : c_2\# \rightarrow c_2\#, \\ &\quad \#_p : c_2p \rightarrow c_2\#, \#_q : c_2q \rightarrow c_2\#, \#_r : c_2r \rightarrow c_2\#\}. \end{aligned}$$

In “state” p we double the number of symbols a_1 , and in “state” q we rename them back from a_2 to a_1 ; in “state” r we rename every symbol a_2 to the output symbol a_3 . The trap rules $\#_p, \#_q, \#_r$ guarantee that the second rule associated with the catalyst c_1 is only carried out with the right state symbol evolving with c_2 . The system in total has 16 rules. We can avoid the trap rules $\#_p, \#_q, \#_r, \#\#$ by using toxic objects, in fact exactly $p, q, r, \#$, thus obtaining a system with only 12 rules:

$$\begin{aligned} \Pi'_{11} &= (O, C = \{c_1, c_2\}, \mu = []_1, w_1 = c_1 c_2 p a_1, R'_1, H', W', f = 1) \text{ where} \\ O &= \{c_1, c_2\} \cup \{a_1, a_2, a_3, p, q, r\}, \\ O_{tox} &= \{p, q, r, \#\}, \\ H' &= \{i \mid 1 \leq i \leq 7\} \cup \{i' \mid i \in \{1, 3, 6\}\} \cup \{\#_{a_1}, \#_{a_2}\}, \\ W' &= \{\{1, 1'\}, \{2, \#_{a_1}\}, \{3, 3'\}, \{4, \#_{a_2}\}, \{5, \#_{a_1}\}, \{6, 6'\}, \{7, \#_{a_2}\}\}, \\ R'_1 &= \{1 : c_2 p \rightarrow c_2 p, 1' : c_1 a_1 \rightarrow c_1 a_2 a_2, 2 : c_2 p \rightarrow c_2 q, \\ &\quad 3 : c_2 q \rightarrow c_2 q, 3' : c_1 a_2 \rightarrow c_1 a_1, 4 : c_2 q \rightarrow c_2 p, \\ &\quad 5 : c_2 p \rightarrow c_2 r, 6 : c_2 r \rightarrow c_2 r, 6' : c_1 a_2 \rightarrow c_1 a_3, \\ &\quad 7 : c_2 r \rightarrow c_2, \#_{a_1} : c_1 a_1 \rightarrow c_1 \#, \#_{a_2} : c_1 a_2 \rightarrow c_1 \#\}. \end{aligned}$$

In sum, we have $mL_{gen, maxpar}(\Pi_{11}) = mL_{gen, maxpar}(\Pi'_{11}) = \{a_3^{2^n} \mid n \geq 0\}$, hence,

$$\{2^n \mid n \geq 1\} \in N_{gen, maxpar} OP_1(pcat_2, ls)_{16} \cap N_{gen, maxpar} O_{tox} P_1(pcat_2, ls)_{12}.$$

Computational completeness of purely catalytic P systems is shown in [12]. We here recall this proof for the generating case, but also show how toxic objects influence the descriptonal complexity of the constructed P system:

Theorem 3. $PsRE = PsOP_1(pcat_2, ls) = PsO_{tox} P_1(pcat_2, ls)$.

Proof. We first prove the inclusion $PsRE \subseteq PsOP_1(pcat_2, ls)$. Let us consider a deterministic register machine $M = (2 + d, B, l_0, l_h, I)$ with the last d registers containing the output values, and let $A = \{a_1, \dots, a_{d+2}\}$ be the set of objects for representing the contents of the registers 1 to $d + 2$ of M . We construct the following purely catalytic P system:

$$\begin{aligned} \Pi &= (O, \{c_1, c_2\}, []_1, c_1 c_2 d l_0, R_1, H, W, 0), \\ O &= A \cup B \cup \{c_1, c_2, d, \#\}, \\ H &= \{l, l', \#_l \mid l \in B\} \cup \{\#_\alpha \mid \alpha \in \{d, \#\}\} \cup \{l_r^-, \#_{a_r} \mid 1 \leq r \leq 2\}; \end{aligned}$$

the sets of labels in W and the rules for R_1 are defined as follows:

A. The trap rules are $\#\# : c_2 \# \rightarrow c_2 \#$ and $\#_d : c_1 d \rightarrow c_1 \#$ as well as $\#_l : c_2 l \rightarrow c_2 \#$ for all $l \in B$ and $\#_{a_r} : c_1 a_r \rightarrow c_1 \#$ for all $a_r, 1 \leq r \leq 2$. Moreover, for every $l \in B$ we define

$$B_\#(l') = \{\#_l \mid l \in B \setminus \{l'\}\}.$$

B. Let $l_i : (\text{ADD}(r), l_j, l_k)$ be an ADD-instruction in I . Then we introduce the two rules

$$l_i : c_2 l_i \rightarrow c_2 l_j a_r \text{ and } l'_i : c_2 l_i \rightarrow c_2 l_k a_r$$

and define $\{l_i, l'_i\}$ to be the corresponding set of labels in W . Observe that only one of these rules can be applied at the same computation step in Π . For the output registers r , $2 < r \leq d + 2$, we replace a_r by (a_r, out) .

C. The simulation of a SUB-instruction $l_i : (\text{SUB}(r), l_j, l_k)$, for $1 \leq r \leq 2$, is carried out by the following rules and the corresponding sets of labels in W :

For the case that the register r is not empty we take $\{l_i, l_r^-, \#_d\} \cup B_{\#}(l_i)$ into W where

$$l_i : c_2 l_i \rightarrow c_2 l_j \text{ and } l_r^- : c_1 a_r \rightarrow c_1.$$

If no symbol a_r is present, i.e., if the register r is empty, then the trap symbol $\#$ is introduced by enforcing c_1 to be used with $\#_d : c_1 d \rightarrow c_1 \#$; moreover, this set of labels should only be used in the presence of l_i as otherwise c_2 is enforced to be used with $\#_l : c_2 l \rightarrow c_2 \#$ for the current label l just being present in the system.

For the case that the register r is empty, we take $\{l'_i, \#_{a_r}\} \cup B_{\#}(l_i)$ into W with the rule

$$l'_i : c_2 l_i \rightarrow c_2 l_k.$$

If at least one symbol a_r is present, i.e., if the register r is not empty, then the trap symbol $\#$ is introduced by the enforced application of the rule $\#_{a_r} : c_1 a_r \rightarrow c_1 \#$; again this set of labels should only be used in the presence of l_i as otherwise c_2 is enforced to be used with $\#_l : c_2 l \rightarrow c_2 \#$ for the current label l just being present in the system.

In both cases, the simulation of the SUB-instruction works correctly if we have made the right choice with respect to the current label present in the system and the contents of register r .

D. At the end of a successful computation of the register machine M , all registers r , $1 \leq r \leq d + 2$, are empty, and M has reached label l_h . Hence, we finally add $\{l_h, l'_h\} \cup B_{\#}(l_h)$ to W where $l_h : c_2 l_h \rightarrow c_2$ and $l'_h : c_1 d \rightarrow c_1$. If during the simulation of the instructions of M by Π no trap symbol $\#$ has been generated, the P system Π halts with only the catalysts remaining in the skin region; otherwise, the system enters an infinite loop with the trap rule $\#_{\#} : c_2 \# \rightarrow c_2 \#$ and $\{\#_{\#}\}$ in W .

In sum, we have shown $L(M) = P_{S_{gen, maxpar}}(\Pi)$, which observation completes the first part of the proof.

We now prove the inclusion $PsRE \subseteq PsO_{tox}P_1(pcat_2, ls)$ and construct the following purely catalytic P system with toxic objects using the rules already

explained above:

$$\begin{aligned}
\Pi' &= (O', \{c_1, c_2\}, []_1, c_1 c_2 d l_0, R'_1, H', W', 0), \\
O' &= A \cup B \cup \{c_1, c_2, d, \#\}, \\
O_{tox} &= B \cup \{\#\}, \\
H' &= \{l, l' \mid l \in B\} \cup \{l_r^- \mid l_i : (\text{SUB}(r), l_j, l_k) \in I\} \cup \{\#\alpha \mid \alpha \in \{d, a_1, a_2\}\}, \\
W' &= \{\{l_h, l'_h\}\} \cup \{\{l_i, l'_i\} \mid l_i : (\text{ADD}(r), l_j, l_k) \in I\} \\
&\quad \cup \{\{l_i, l_r^-, \#d\}, \{l'_i, \#a_r\} \mid l_i : (\text{SUB}(r), l_j, l_k) \in I\}, \\
R'_1 &= \{l_i : c_2 l_i \rightarrow c_2 l_j a_r, l'_i : c_2 l_i \rightarrow c_2 l_k a_r \mid l_i : (\text{ADD}(r), l_j, l_k) \in I, 1 \leq r \leq 2\} \\
&\quad \cup \{l_i : c_2 l_i \rightarrow c_2 l_j (a_r, out), l'_i : c_2 l_i \rightarrow c_2 l_k (a_r, out) \mid \\
&\quad \quad l_i : (\text{ADD}(r), l_j, l_k) \in I, 2 < r \leq d + 2\} \\
&\quad \cup \{l_i : c_2 l_i \rightarrow c_2 l_j, l_r^- : c_1 a_r \rightarrow c_1, l'_i : c_2 l_i \rightarrow c_2 l_k \mid l_i : (\text{SUB}(r), l_j, l_k) \in I\} \\
&\quad \cup \{\#\alpha : \alpha \rightarrow \# \mid \alpha \in \{d, a_1, a_2\}\} \cup \{l_h : c_2 l_h \rightarrow c_2, l'_h : c_1 d \rightarrow c_1\}.
\end{aligned}$$

In this purely catalytic P system with toxic objects, besides the trap symbol itself, exactly the labels from B are toxic, i.e., they must evolve, which guarantees that a set from W' is used with the correct label. This observation concludes the proof showing $L(M) = P_{s_{gen, maxpar}}(\Pi')$. \square

4.6 Generating Number Sets with [Purely] Catalytic P Systems

We now are going to improve the result from [21], where a catalytic P system with 54 rules was elaborated, generating the non-semilinear set of natural numbers $\{2^n - 2n \mid n \geq 2\}$, and even the improved result from [22] where only 32 rules were needed. In the following, we construct a [purely] catalytic P system with 29 rules, generating the (standard) non-semilinear set of natural numbers $\{2^n \mid n \geq 1\}$. Yet we will show even more, i.e., our construction works for any set of natural numbers representing a function $g : \mathbb{N} \rightarrow \mathbb{N}$ which is computed in r_3 by the following function program (starting with $r_1 = b_0$ and $r_2 = r_3 = 0$), with r_1, r_2, r_3 representing three registers of a register machine, and with the parameters b_0, b_1, b_2, b_3, b_4 being natural numbers, $b_0 \geq 1$:

```

function  $g(r_3)$ :
1: if  $r_1 > 0$ 
    then begin DEC( $r_1$ ); ADD(1,  $r_2$ ); goto 1 end
    else goto 1
    orelse begin ADD( $b_3, r_3$ ); goto 3 end
2: if  $r_2 > 0$ 
    then begin DEC( $r_2$ ); ADD( $b_2, r_1$ ); ADD( $b_1, r_3$ ); goto 2 end
    else begin ADD( $b_4, r_1$ ); goto 1 end;
3: HALT
endfunction

```

The idea of this program is that in label 1 we copy (register) r_1 to r_2 ; the notation DEC(r) means decrementing (register) r by one, whereas ADD(k, r)

means adding k to (register) r . As soon as register r_1 is empty, we switch to label 2 or halt after having added b_3 to (the result register) r_3 before. In label 2, we copy back the value of register r_2 to r_1 , but take it b_2 times, at the same time adding b_1 times the value of register r_2 to r_3 , and at the end, when r_2 is empty, we add b_4 to r_1 .

The structures

```
i: if  $r_i > 0$ 
      then begin DEC( $r_i$ ); ADD( $1, r_{3-i}$ ); goto i_end
      else goto j
```

in this function program correspond with the following instructions in a register machine program:

```
 $i$  : (SUB( $i, i', j$ ))
 $i'$  : (ADD( $3 - i, i, i$ ))
```

We now describe the functions computed by specific values of the parameters b_0, b_1, b_2, b_3, b_4 ; thereby let $f_i(n)$, $i \in \{1, 3\}$, denote the value of register i after n times, $n \geq 0$, having gone through the loops 1 and 2, and $g(n)$ the final value of the function when going through the loops 1 and 2 for n times and then performing loop 1 once more, yet exiting at the end of loop 1 to halt.

In general, we get $f_2(0) = f_2(n) = 0$ for all $n \geq 0$ as well as the system of linear recursions

$$\begin{aligned} f_1(n+1) &= b_2 f_1(n) + b_4, \\ f_3(n+1) &= f_3(n) + b_1 f_1(n) \end{aligned}$$

with $f_1(0) = b_0$ and $f_3(0) = 0$ as well as the final result $g(n) = f_3(n) + b_3$.

Case 1. $b_2 = 1$:

In this case, we get the system of linear recursions

$$\begin{aligned} f_1(n+1) &= f_1(n) + b_4, \\ f_3(n+1) &= f_3(n) + b_1 f_1(n). \end{aligned}$$

Solving these recursions yields $f_1(n) = b_0 + b_4 n$ and, for $n \geq 0$,

$$\begin{aligned} f_3(n) &= f_3(0) + \sum_{i=0}^{n-1} b_1 f_1(i) = b_1 \sum_{i=0}^{n-1} (b_0 + b_4 i) \\ &= b_1 b_0 n + b_1 b_4 n(n-1)/2 \end{aligned}$$

hence, $g(0) = b_3$ and, for $n \geq 0$,

$$g(n) = f_3(n) + b_3 = (b_1 b_4 / 2) n^2 + (b_1 b_0 - b_1 b_4 / 2) n + b_3,$$

i.e., a quadratic function provided $b_1 \neq 0$ and $b_4 \neq 0$.

As a specific example, for $(b_0, b_1, b_2, b_3, b_4) = (1, 1, 1, 0, 2)$ we obtain $g(n) = n^2$.

Case 2. $b_2 > 1$, $b_1 = 1$, $b_4 = 0$:

In this case, we get the linear recursions

$$\begin{aligned} f_1(n+1) &= b_2 f_1(n), \\ f_3(n+1) &= f_3(n) + f_1(n) \end{aligned}$$

with $f_1(0) = b_0$ and $f_2(0) = f_3(0) = 0$ as well as the final result $g(n) = f_3(n) + b_3$, i.e., for $n \geq 0$ we obtain $f_1(n) = b_0(b_2)^n$ and

$$f_3(n) = f_3(0) + \sum_{i=0}^{n-1} b_0(b_2)^i = b_0 \sum_{i=1}^{n-1} (b_2)^i = b_0(((b_2)^n - 1)/(b_2 - 1))$$

as well as

$$g(n) = f_3(n) + b_3 = b_0(((b_2)^n - 1)/(b_2 - 1)) + b_3.$$

As a specific example, for $(b_0, b_1, b_2, b_3, b_4) = (1, 1, 2, 1, 0)$ we therefore obtain $g(n) = 2^n$.

We now start from the constructions for simulating SUB-instructions as already exhibited in Subsection 4.3. In order to get even more efficient simulations, we save the first steps in a specific way; moreover, every ADD-instruction can be incorporated into the rules of the last steps of the simulations, in a similar way as this was already done in the construction elaborated in [21]. Thus, for the function g with the parameters b_0, b_1, b_2, b_3, b_4 we construct the catalytic P system

$$\begin{aligned} \Pi_{12}(b_0, b_1, b_2, b_3, b_4) &= (O, C = \{c_1, c_2\}, \mu = [\]_1, w_1, R_1, f = 1) \text{ where} \\ O &= \{a_1, a_2, a_3, c'_1, c'_2, d, \#\} \\ &\cup \{p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1, 2\}\}, \\ w_1 &= c_1 c_2 (a_1)^{b_0} p_1, \\ R_1 &= R_{1,c} \cup R_{1,\#}, \end{aligned}$$

and $R_{1,c}$ consists of the catalytic rules contained in the following two tables:

Simulation of the instructions related with label 1 if register 1 is not empty	register 1 is empty
$c_1 a_1 \rightarrow c_1 c'_1$	c_1 remains idle
$c_2 p_1 \rightarrow c_2 p'_1$	$c_2 \bar{p}_1 \rightarrow c_2 p_2$
$c_1 c'_1 \rightarrow c_1 d$	
$c_2 p'_1 \rightarrow c_2 p''_1$	
$c_1 p''_1 \rightarrow c_1 p_1 a_2$ or	
$c_1 p''_1 \rightarrow c_1 \bar{p}_1 a_2$	
$c_2 d \rightarrow c_2$	
halting:	$c_2 \bar{p}_2 \rightarrow c_2 (a_3)^{b_3}$

Simulation of the instructions related with label 2 if register 2 is not empty	register 1 is empty
$c_2 a_2 \rightarrow c_2 c'_2$	c_2 remains idle
$c_1 p_2 \rightarrow c_1 p'_2$	$c_1 \bar{p}_2 \rightarrow c_1 p_1 (a_1)^{b_4}$
$c_2 c'_2 \rightarrow c_2 d$	
$c_1 p'_2 \rightarrow c_1 p''_2$	
$c_2 p''_2 \rightarrow c_2 p_2 (a_1)^{b_2} (a_3)^{b_1}$ or	
$c_2 \bar{p}''_2 \rightarrow c_2 \bar{p}_2 (a_1)^{b_2} (a_3)^{b_1}$	
$c_1 d \rightarrow c_1$	

In addition, we have to add trap rules to guarantee that in case of wrong guesses, the derivation enters an infinite loop with the rule $\# \rightarrow \#$ in the catalytic case (or $c_3 \# \rightarrow c_3 \#$ in the purely catalytic case). The objects x for which we have such trap rules $x \rightarrow \#$ in the catalytic case (or $c_3 x \rightarrow c_3 \#$ in the purely catalytic case) are $\#$ and d as well as, for $j \in \{1, 2\}$, the objects $c'_j, p_j, p'_j, p''_j, \bar{p}_j$, i.e.,

$$R_{1,\#} = \{x \rightarrow \# \mid x \in \{\#, d\} \cup \{c'_j, p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1, 2\}\}\}.$$

In total this yields 17 catalytic rules in $R_{1,c}$ and 12 trap rules in $R_{1,\#}$, i.e., 29 rules in R_1 . Obviously, the same number of rules is obtained for the corresponding purely catalytic P system $\Pi'_{12}(b_0, b_1, b_2, b_3, b_4)$ where we simply have to add the third catalyst c_3 and replace the context-free trap rules $x \rightarrow \#$ by the corresponding catalytic trap rules $c_3 x \rightarrow c_3 \#$.

We can omit the trap rules when using toxic objects, i.e., if we take

$$\begin{aligned} \Pi''_{12}(b_0, b_1, b_2, b_3, b_4) &= (O'', C = \{c_1, c_2\}, O_{tox}, \mu = [\]_1, w_1, R_{1,c}, f = 1) \text{ where} \\ O'' &= \{a_1, a_2, a_3, c'_1, c'_2, d\} \\ &\quad \cup \{p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1, 2\}\}, \\ O_{tox} &= \{c'_1, c'_2, d\} \cup \{p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1, 2\}\}, \\ w_1 &= c_1 c_2 (a_1)^{b_0} p_1, \end{aligned}$$

and $R_{1,c}$ contains the catalytic rules as listed above. This system now only contains 17 rules.

How to argue that the catalytic P system $\Pi_{12}(b_0, b_1, b_2, b_3, b_4)$ and the corresponding catalytic P systems $\Pi'_{12}(b_0, b_1, b_2, b_3, b_4)$ and $\Pi''_{12}(b_0, b_1, b_2, b_3, b_4)$ work correctly was exhibited in detail in [10] as well as in [21] and [22]. Yet as we have reduced the number of rules in a considerable way, we have to argue for any possible case of decrementing or zero-test register a :

If decrementing of register a is possible, all steps have to be performed exactly as described in the table. If decrementing fails, then in the last (third) step c_{3-a} must be used with the rule $c_{3-a} a_{3-a} \rightarrow c_{3-a} c'_{3-a}$ as not both registers can be empty during a computation in the register machine. Yet in the next step the catalyst c_{3-a} is busy with the program symbol p_a or \bar{p}_a , hence, with c'_{3-a} and one of these program symbols competing for the same catalyst, one of these symbols will be trapped.

A successful simulation of testing register a for zero is performed in one step leaving catalyst c_a idle. In case the register is not empty, c'_a has to be generated, and this symbol in the next step will compete for the catalyst c_a with the program symbol p_{3-a} and thus one of these symbols will be trapped.

As already explained in [21], we here also mention that when using $c_2\bar{p}_2 \rightarrow c_2(a_3)^{b_3}$ instead of $c_2\bar{p}_1 \rightarrow c_2p_2$ in order to reach a halting configuration, the system does not immediately halt, but instead, if having chosen the rule when register 1 is empty, uses the sequences of rules $c_2a_2 \rightarrow c_2c'_2$, $c_2c'_2 \rightarrow c_2d$, and $c_1d \rightarrow c_1$ (or $c_1d \rightarrow c_1$), to clear register 2, so that in the end only the objects a_3 remain besides the catalysts. If $c_2\bar{p}_2 \rightarrow c_2(a_3)^{b_3}$ is chosen too early, then both registers may be cleared by using the corresponding rules. The result expressed by the number of symbols a_3 is not affected, if we make such a wrong choice at the end only.

As specific examples, we therefore obtain

$$N_{gen,maxpar}(\Pi_{12}(1, 1, 1, 0, 2)) = \{n^2 \mid n \geq 0\}$$

and

$$N_{gen,maxpar}(\Pi_{12}(1, 1, 2, 1, 0)) = \{2^n \mid n \geq 1\}.$$

We observe that with respect to the complexity of the systems, especially concerning the number of rules, there is no difference at all between the sets of natural numbers growing in a quadratic and in an exponential way, respectively.

In sum, we conclude that all these non-linear sets of natural numbers as described above are contained in

$$N_{gen,maxpar}OP_1(cat_2)_{29} \cap N_{gen,maxpar}OP_1(pcat_3)_{29}$$

as well as in

$$N_{gen,maxpar}O_{tox}P_1(cat_2)_{17} \cap N_{gen,maxpar}O_{tox}P_1(pcat_2)_{17}.$$

If we do not limit ourselves with the number of catalysts, a better solution with respect to the number of rules is possible, i.e., for the function g with the parameters b_0, b_1, b_2, b_3, b_4 we construct the purely catalytic P system

$$\begin{aligned} \Pi_{13}(b_0, b_1, b_2, b_3, b_4) &= (O, C, =, \mu = []_1, w_1, R_1, f = 1) \text{ where} \\ O &= C \cup \{a_1, a_2, a_3, p_1, p_2, p_h, d_1, d_2, d'_1, d'_2, d, d', \#\} \\ C &= \{c_{r,Decr}, c_{r,0Test} \mid r \in \{1, 2\}\} \cup \{c_d, c_p, c_\#\} \\ w_1 &= c_{1,Decr}c_{1,0Test}c_{2,Decr}c_{2,0Test}c_dc_p c_\#(a_1)^{b_0}p_1d_2d'_1d'_2dd', \end{aligned}$$

and R_1 consists of the catalytic rules described in the following.

The main idea of this new construction is to use two catalysts for each register – one for the decrement ($c_{r,Decr}$) and one for the zero-test ($c_{r,0Test}$). Moreover, each SUB-instruction is simulated by two rules, one for the decrement and one for the zero-test, just allowing the corresponding catalyst to do its work, whereas all other catalysts are kept busy having introduced d_r for $c_{r,Decr}$ and d'_r for $c_{r,0Test}$.

The catalyst c_d for the special symbol d is kept busy by d' ; the symbol d is used for trapping in case an intended decrement fails and can only be allowed to vanish in the last step. The catalyst c_p is used with the instruction labels p_1 , p_2 , and p_h . The catalyst $c_\#$ is only needed for handling the trap symbol $\#$.

For each of the two registers $r \in \{1, 2\}$, the following rules perform the decrement and the zero-test, respectively, in case this operation is initiated by omitting d_r or d'_r , respectively, in the step before.

decrement $c_{r,Decr}a_r \rightarrow c_{r,Decr}$: if register r is not empty, it is decremented;
 $c_{r,Decr}d \rightarrow c_{r,Decr}\#$: if register r is empty, the catalyst $c_{r,Decr}$ has to be used with the symbol d thereby introducing the trap symbol $\#$;
 $c_{r,Decr}d_r \rightarrow c_{r,Decr}$: d_r keeps $c_{r,Decr}$ busy if another instruction is to be simulated;
 $c_\#d_r \rightarrow c_\#\#$: d_r is a “toxic” object which must not stay idle;
 $c_d d' \rightarrow c_d$: d' keeps c_d busy until the end;
 $c_\# d' \rightarrow c_\#\#$: d' is a “toxic” object which must not stay idle.
zero-test $c_{r,0Test}a_r \rightarrow c_{r,0Test}\#$: if register r is not empty, a trap symbol $\#$ is generated;
 $c_{r,0Test}d'_r \rightarrow c_{r,0Test}$: d'_r keeps $c_{r,0Test}$ busy if another instruction is to be simulated; for the symbol d'_r we do not need an additional trap rule as the only alternative is already a trap rule.

The following rules initiate the decrement or the zero-test on register 1 or 2 and simulate the program for the function g :

1. $c_p p_1 \rightarrow c_p a_2 p_1 d_2 d'_1 d'_2 d'$: decrement register 1;
 $c_p p_1 \rightarrow c_p p_2 d_1 d_2 d'_1 d'_2 d'$: zero-test register 1;
2. $c_p p_2 \rightarrow c_p (a_1)^{b_2} (a_3)^{b_1} p_2 d_1 d'_1 d'_2 d'$: decrement register 2;
 $c_p p_2 \rightarrow c_p (a_1)^{b_4} p_1 d_1 d_2 d'_1 d'$: zero-test register 2;
 $c_p p_1 \rightarrow c_p (a_3)^{b_3} p_h d_1 d_2 d'_2 d'$: zero-test register 1 and go to halting.
3. Whereas register 1 is already empty, now also register 2 has to be cleaned using the instruction label p_h :
 $c_p p_h \rightarrow c_p p_h d_1 d'_1 d'_2 d'$: decrement register 2;
 $c_p p_h \rightarrow c_p d_1 d_2 d'_1$: zero-test register 2 and eliminate p_h ;
 $c_d d \rightarrow c_d$: finally c_d is allowed to eliminate d ;
 $c_\# \# \rightarrow c_\#\#$: in case something goes wrong during a simulation of an instruction, this rule keeps the P system in an infinite loop.

In sum, this purley catalytic P system $\Pi_{13}(b_0, b_1, b_2, b_3, b_4)$ contains only 23 rules. We can save two catalysts by using non-cooperative rules instead of the catalytic rules assigned to the catalysts c_p and $c_\#$, thus obtaining the catalytic P system $\Pi'_{13}(b_0, b_1, b_2, b_3, b_4)$. Hence, all the non-linear sets of natural numbers described above are contained in

$$N_{gen,maxpar}OP_1(cat_5)_{23} \cap N_{gen,maxpar}OP_1(pcat_7)_{23}.$$

Besides the trap rule $c_\#\# \rightarrow c_\#\#$, only the rules $c_\#d_r \rightarrow c_\#\#$, $r \in \{1, 2\}$, and $c_\#d' \rightarrow c_\#\#$ can be omitted when considering a (purely) catalytic P system with “toxic” objects, yet this result with 19 rules is even weaker than the previous one where we also used less catalysts.

5 Conclusions

In this paper we have investigated and illustrated with several examples the effect of using toxic objects in various models of P systems. Moreover, we have given a lot of examples for small P systems accepting or generating specific non-semilinear sets of vectors of natural numbers or non-semilinear sets of natural numbers. As our main result, we have improved considerably the result established in [21] and even improved the newest result obtained in [22] by showing that 29 rules are enough for generating the non-semilinear set of numbers $\{2^n \mid n \geq 1\}$ with (purely) catalytic P systems and 2 (3) catalysts; using toxic objects, only 17 rules are needed. Allowing for a larger number of catalysts, with a new proof technique we could even reduce the number of rules to 23.

For the catalytic P systems/purely catalytic P systems it is still one of the most challenging questions in the area of P systems whether we really need two/three catalysts to get computational completeness or at least to accept or generate a non-semilinear set of (vectors of) natural numbers. Another direction for future research is to investigate the influence of toxic objects in further models of P systems.

Acknowledgements

The first author acknowledges project STCU-5384 *Models of high performance computations based on biological and quantum approaches* awarded by the Science and Technology Center in the Ukraine. Both authors are very grateful to Petr Sosik for pointing out the mistake in the proof of Corollary 8 in [10].

References

1. A. Alhazov, R. Freund: Asynchronous and Maximally Parallel Deterministic Controlled Non-cooperative P Systems Characterize NFIN and coNFIN. In: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil (Eds.): *Membrane Computing - 13th International Conference, CMC 2012*, Budapest, Revised Selected Papers. *Lecture Notes in Computer Science* **7762**, Springer, 2013, 101–111.
2. A. Alhazov, R. Freund: Small P Systems Defining Non-semilinear Sets. *To appear*.
3. A. Alhazov: P Systems without Multiplicities of Symbol-Objects. *Information Processing Letters* **100** (3), 124–129 (2006).
4. A. Alhazov, R. Freund, Gh. Păun: Computational Completeness of P Systems with Active Membranes and Two Polarizations. In: M. Margenstern (Ed.): *Machines, Computations, and Universality, International Conference, MCU 2004*, Saint Petersburg, Revised Selected Papers. *Lecture Notes in Computer Science* **3354**, Springer, 2005, 82–92.
5. A. Alhazov, R. Freund, A. Riscos-Núñez: One and Two Polarizations, Membrane Creation and Objects Complexity in P Systems. *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 05*. IEEE Computer Society, 2005, 385–394.
6. A. Alhazov, S. Verlan: Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. *Theoretical Computer Science* **412** (17), 1581–1591 (2011).

7. M. Beyreder, R. Freund: Membrane Systems Using Noncooperative Rules with Unconditional Halting. *Membrane Computing. 9th International Workshop, WMC 2008*, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science **5391**, Springer, 2009, 129–136. DOI=10.1007/978-3-540-95885-7_10.
8. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
9. R. Freund: Special Variants of P Systems Inducing an Infinite Hierarchy with Respect to the Number of Membranes. *Bulletin of the EATCS* **75**, 209–219 (2001).
10. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science* **330** (2), 251–266 (2005).
11. R. Freund, A. Leporati, G. Mauri, A. E. Porreca, S. Verlan, C. Zandron: Flattening in (Tissue) P Systems. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, A. Salomaa (Eds.): *Membrane Computing – 14th International Conference, CMC 2013*, Chişinău, Republic of Moldova, August 20–23, 2013, Revised Selected Papers. Lecture Notes in Computer Science **8340**, Springer 2014, 173–188. DOI=10.1007/978-3-642-54239-8.
12. R. Freund, M. Oswald, Gh. Păun: Catalytic and Purely Catalytic P Systems and P Automata: Control Mechanisms for Obtaining Computational Completeness. *Fundamenta Informaticae*, to appear.
13. O.H. Ibarra, S. Woodworth: On Symport/Antiport P Systems with a Small Number of Objects. *International Journal of Computer Mathematics* **83** (7), 2006, 613–629. 137–152.
14. M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
15. Gh. Păun: Computing with Membranes. *J. Comput. Syst. Sci.* **61**, 108–143 (2000); also see TUCS Report 208, 1998, www.tucs.fi.
16. Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
17. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010, 118–143.
18. G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
19. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*, 3 volumes. Springer, 1997.
20. D. Sburlan: Further Results on P Systems with Promoters/Inhibitors. *International Journal of Foundations of Computer Science* **17** (1), 205–221 (2006).
21. P. Sosík: A Catalytic P System with Two Catalysts Generating a Non-Semilinear Set. *Romanian Journal of Information Science and Technology* **16** (1), 3–9 (2013).
22. P. Sosík, M. Langer: Improved Universality Proof for Catalytic P Systems and a Relation to Non-Semi-Linear Sets. In: S. Bensch, R. Freund, F. Otto (Eds.): *Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA 2014)*, books@ocg.at, Band **304**, Wien, 2014, 223–234.
23. The P systems webpage. <http://ppage.psyste.ms.eu>

Length P Systems with a Lonesome Traveler

Artiom Alhazov¹, Rudolf Freund², and Sergiu Ivanov³

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

² Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
E-mail: rudi@emcc.at

³ LACL, Université Paris Est – Créteil Val de Marne
61, av. Général de Gaulle, 94010, Créteil, France
Email: sergiu.ivanov@u-pec.fr

Abstract. In this paper we examine P systems with linear membrane structures (only one membrane is elementary), in most cases considering systems with at most one object. We prove that two labels, elementary membrane creation and dissolution, together with the usual send-in and send-out rules, suffice to achieve computational completeness, even with the restriction that only one object is allowed to be present in any configuration of the system. On the other hand, limiting the number of labels to one reduces the computational power to regular sets of non-negative integers. We also consider the possibility of interpreting the sequences of membrane labels in halting configurations as languages and we prove that all *RE* languages can be generated in this way. Finally, we show that, when multiple objects can appear in the system, it suffices to only allow one membrane to be labeled differently from the other membranes to achieve computational completeness.

1 Introduction

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

Unbounded linear membrane structures have received the attention of researchers in the past, see, e.g., [5] and [4]. In the latter paper, the authors discussed the generation of *languages* by representing strings $a_1 \cdots a_n$ as membrane labels arranged in a linear structure as follows:

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

A different example of research where unbounded membrane structures played a crucial role for obtaining an important result – the computational completeness of P systems with active membranes without polarizations – is given in [1], improved in terms of presentation and object/symbol/membrane label complexity in [3].

This research direction, focusing on the membrane structure – rather than the multiset of objects in a designated region – as the result of the computation of a P system, has been recalled during the 12th Brainstorming Week on Membrane Computing in Sevilla, see [2]. The technique proposed there of how to generate (the description of) recursively enumerable sets of vectors of non-negative integers, using membrane structures with only two labels (0 and 1), where the number of membranes labeled by 1 remains bounded by a constant throughout the computation, is explained in Section 3. In Section 6 we drop this restriction, thus generating languages. It was also conjectured that:

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

We confirm this conjecture (referred to as “the regularity conjecture”) in Section 4. Finally, in Section 5 we discuss variants of the model leading to weak computational completeness, while in Section 7 we strengthen this result by considering multiple objects.

2 Definitions

We assume the reader to be familiar with the basic notions and results of formal language theory, e.g., see [11], and of membrane systems, see the monographs [9] and [10] as well as the P systems webpage [12].

The families of regular and recursively enumerable string languages are denoted by *REG* and *RE*, respectively. The corresponding families of sets of numbers as well as of vectors of numbers are denoted by *NREG* and *NRE* as well as by *PsREG* and *PsRE*, respectively. The languages generated by matrix grammars are denoted by *MAT*. The corresponding families of sets of numbers are denoted by *NMAT*.

A *regular* (right-linear) *grammar* is a construct $G = (N, T, S, P)$, where N is the set of nonterminals, T is the set of terminals, S is the start symbol, and P is a set of rules of the form $A \rightarrow bC$ and $A \rightarrow \lambda$ with $A, C \in N$ and $b \in T$ (λ is the empty string). The language generated by G is defined as the set of words over T which can be obtained from S by sequentially applying the rules from P . Without loss of generality we may assume that G is *reduced*, i.e., every nonterminal in N is reachable from S and from every nonterminal $A \in N$ (eventually except S , if the empty set is generated by the set of rules $\{S \rightarrow aS\}$) a terminal string can be derived. The family of languages generated by (reduced) regular grammars exactly coincides with *REG*.

A *linear* grammar is a construct $G = (N, T, S, P)$, where N , T , and S are defined as above, while P is a set of productions of the form $A \rightarrow \alpha$, where

$A \in N$ and w is a string over $N \cup A$ such that it contains at most one symbol from N . The language generated by G is defined in the same way as in the case of regular grammars. A language is called linear, if it is generated by a linear grammar.

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

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

A *configuration* of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. Computations start by executing the instruction l_0 of P , and terminate with reaching the HALT-instruction l_h . We say that \mathcal{M} computes the output values y_1, \dots, y_k , $y_j \in \mathbb{N}$, $1 \leq j \leq k$, on the input x_1, \dots, x_n , $x_i \in \mathbb{N}$, $1 \leq i \leq n$, if, starting from the initial configuration in which the first n registers of \mathcal{M} contain the values x_i , the machine reaches the HALT-instruction with the values y_j in its first k registers.

Register machines are well-known computationally complete mechanisms, e.g., see [8].

2.1 P Systems with Membrane Creation and Dissolution

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

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

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

Each rule in R is of one of the following types (we avoid the label (a) because it is usually reserved for evolution rules):

- (b) $[_{h_2} a[_{h_1}]_{h_1}]_{h_2} \rightarrow [_{h'_2} [_{h'_1} b]_{h'_1}]_{h'_2}$,
 $a, b \in O, h_1, h_2, h'_1, h'_2 \in H$ – *send-in* rule,
- (c) $[_{h_2} [_{h_1} a]_{h_1}]_{h_2} \rightarrow [_{h'_2} b[_{h'_1}]_{h'_1}]_{h'_2}$,
 $a, b \in O, h_1, h_2, h'_1, h'_2 \in H$ – *send-out* rule,
- (d) $[_{h_2} [_{h_1} a]_{h_1}]_{h_2} \rightarrow [_{h'_2} b]_{h'_2}$,
 $a, b \in O, h_1, h_2, h'_2 \in H$ – *membrane dissolution* rule,
- (e) $[_{h_1} a]_{h_1} \rightarrow [_{h'_1} [_{h_2} b]_{h_2}]_{h'_1}$,
 $a, b \in O, h_1, h_2, h'_1 \in H$ – *membrane creation* rule.

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

A rule of type (e) consumes an instance of a in membrane with label h_1 and adds to it a new membrane with label h_2 , with an instance of b inside. The label of the original membrane is rewritten to h'_1 . Symmetrically, a rule of type (d) consumes an instance of a in a membrane with label h_1 , which is located within a membrane with label h_2 , copies all the symbols from the membrane with label h_1 to its parent membrane, discards the membrane h_1 , adds a b to the membrane h_2 and rewrites its label to h'_2 .

The rules are applied in the maximally parallel way, with the restriction that in one derivation step at most one rule of types (b), (c), (d), and (e) can be applied per each membrane labeled by h_1 in the definition of the rules given above. We deliberately avoid evolution rules, so that no parallelism can happen at the level of any membrane.

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

Note: In the following, we will restrict ourselves to P systems with linear membrane structure. In such systems, membrane creation should never be applied in a non-elementary membrane, otherwise a non-linear membrane structure would be obtained, which is unwanted in the model of the current paper. In Section 3, the P system is constructed in such a way that the object triggering membrane creation would only appear in the elementary membrane. However, in Sections 4 and 5, the setup is more restricted (e.g., one label only), which would yield too restrictive P systems. To overcome this, we impose the following restriction: membrane creation rules are disabled in non-elementary membranes. We refer to this kind of rules as (e_e) . In a similar way, we are interested in membrane

dissolution rules which are disabled in non-elementary membranes, and we denote them by (d_e) . Clearly, for each of these membrane dissolution and creation operations, the constructions in Section 3 work for either variant (emphasizing (d_e) , otherwise the decrement could be simplified). The regularity conjecture originally assumed only elementary dissolution to be used as well as results only to be obtained at halting with the object in the elementary membrane.

In Section 5 we also consider the following type of rule (introduced already in [4]; we have removed the object in the inner membrane on the right side to let the systems considered in the paper have at most one object during the computation):

$$(f) \quad [_{h_1} a]_{h_1} \rightarrow [_{h'_1} b[_{h_2}]_{h_2}]_{h'_1}, \\ a, b \in O, h_1, h_2, h'_1 \in H - \text{membrane duplication rule:}$$

the membrane h_1 , in the presence of an object a , is duplicated, that is, the label h_1 is changed into h'_1 , the object a is replaced by b and a new inner membrane labeled by h_2 is created; all the contents of membrane h_1 (membranes or objects except this copy of object b) is now inside membrane h_2 .

Note: In [4] the authors have assumed the outer membrane to be the newly created one; it makes no difference as long as we can change both labels by this rule. However, we prefer to view the *inner* membrane as the new membrane. This lets us keep $h'_1 = h_1$ whenever we need it.

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

$$(b_r) \quad a[_{h_1}]_{h_1} \rightarrow [_{h'_1} b]_{h'_1}, \quad a, b \in O, h_1, h'_1 \in H, \\ (c_r) \quad [_{h_1} a]_{h_1} \rightarrow b[_{h'_1}]_{h'_1}, \quad a, b \in O, h_1, h'_1 \in H, \\ (d_r) \quad [_{h_1} a]_{h_1} \rightarrow b, \quad a, b \in O, h_1 \in H \text{ (we write } (d_{er}) \text{ if the rule is only applicable for the elementary membrane).}$$

One difference, not relevant for P systems with at most one object, is that the outermost membrane, not reflected in a rule, is no longer bound by this rule, and can simultaneously be used for some other rule with a different object.

3 Length P Systems

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

- the membrane structure is *linear* in every configuration, i.e., every membrane has at most one inner membrane;

- the membranes in any configuration of Π reachable from the initial one are labeled by one out of at most two labels only.

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

Following the same convention, we can define the input of Π as a two-label membrane structure coding a certain vector of numbers, i.e., for an input vector $v = (v_1, v_2, \dots, v_n)$, we construct the initial configuration of the length P system Π with the membranes labeled according to the string $\alpha 10^{v_1} 10^{v_2} 1 \dots 10^{v_n} 1 \beta$ over H , where $\alpha, \beta \in H^*$ are fixed and do not depend on v . For a different example of designing P systems with the membrane structure being a part of the input, see [6].

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

The evolution of the system starts with the rule

$$[{}_h q_s [{}_1]_1]_h \rightarrow [{}_h [{}_1 q_{1,1}]_1]_h, 0 \leq h \leq 1,$$

where the symbol $q_{1,1}$ represents the first instruction in the program of \mathcal{M} and also keeps the information about the fact that it is located in the region of the membrane structure corresponding to the first register of \mathcal{M} . We use the generic label h in this case to show that we do not ever depend on the actual label of the skin (as the traveler can detect that it is in the skin by counting the membranes labeled by 1 it traverses on its way out).

To simulate an increment of the i -th register of \mathcal{M} , we need to add a membrane to the membrane structure and assure that the sequence of labels changes from $110^{R_1} \dots 10^{R_i} \dots 10^{R_m} 1$ to $110^{R_1} \dots 10^{R_i+1} \dots 10^{R_m} 1$. We start with the symbol $q_{l,1}$ in the inner membrane of the skin, where l is the label of the increment operation, and we move it to the innermost membrane, counting the registers we traverse on that way:

$$\begin{aligned} [{}_h q_{l,j} [{}_0]_0]_h &\rightarrow [{}_h [{}_0 q_{l,j}]_0]_h, \\ [{}_h q_{l,j} [{}_1]_1]_h &\rightarrow [{}_h [{}_1 q_{l,j+1}]_1]_h, 1 \leq j \leq m, 0 \leq h \leq 1. \end{aligned}$$

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

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

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

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

$$\begin{aligned} [{}_0 [{}_0 s_{l,j}]_0]_0 &\rightarrow [{}_0 s_{l,j} [{}_0]_0]_0, \\ [{}_1 [{}_0 s_{l,j}]_0]_1 &\rightarrow [{}_0 s_{l,j-1} [{}_1]_1]_0, \quad i < j \leq m+1. \end{aligned}$$

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

$$\begin{aligned} [{}_h [{}_0 s_{l,j}]_0]_h &\rightarrow [{}_h s_{l,j} [{}_0]_0]_h, \\ [{}_h [{}_1 s_{l,j}]_1]_h &\rightarrow [{}_h s_{l,j-1} [{}_1]_1]_h, \quad 1 \leq j \leq i, \quad 0 \leq h \leq 1. \end{aligned}$$

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

$$[{}_h s_{l,0} [{}_1]_1]_h \rightarrow [{}_h [{}_1 q_{l',1}]_1]_h, \quad 0 \leq h \leq 1.$$

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

$$\begin{aligned} [{}_h [{}_0 q_{l,j}]_0]_h &\rightarrow [{}_h [{}_0 q_{l,j}]_0]_h, \\ [{}_h [{}_1 q_{l,j}]_1]_h &\rightarrow [{}_h [{}_1 q_{l,j+1}]_1]_h, \quad 1 \leq j < i, \quad 0 \leq h \leq 1. \end{aligned}$$

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

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

The signal symbol then travels to the skin membrane:

$$\begin{aligned} [{}_h [{}_0 z_{l,j}]_0]_h &\rightarrow [{}_h z_{l,j} [{}_0]_0]_h, \\ [{}_h [{}_1 z_{l,j}]_1]_h &\rightarrow [{}_h z_{l,j-1} [{}_1]_1]_h, \quad 1 \leq j \leq i, \quad 0 \leq h \leq 1. \end{aligned}$$

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

$$[{}_h z_{l,0} [{}_1]_1]_h \rightarrow [{}_h [{}_1 q_{l',1}]_1]_h, \quad 0 \leq h \leq 1.$$

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

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

This symbol moves all membrane labels one step outwards:

$$\begin{aligned} [{}_0 d_{l,j} [{}_0]_0]_0 &\rightarrow [{}_0 [{}_0 d_{l,j}]_0]_0, & i \leq j \leq m, \\ [{}_0 d_{l,j} [{}_1]_1]_0 &\rightarrow [{}_1 [{}_0 d_{l,j+1}]_0]_1, & i \leq j \leq m. \end{aligned}$$

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

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

Now the s -symbol goes outwards to the skin membrane:

$$\begin{aligned} [{}_h [{}_0 s_{l,j}]_0]_h &\rightarrow [{}_h s_{l,j} [{}_0]_0]_h, \\ [{}_h [{}_1 s_{l,j}]_1]_h &\rightarrow [{}_h s_{l,j-1} [{}_1]_1]_h, & 1 \leq j \leq m+1, & 0 \leq h \leq 1. \end{aligned}$$

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

$$[{}_h s_{l,0} [{}_1]_1]_h \rightarrow [{}_h [{}_1 q''_{,1}]_1]_h, & 0 \leq h \leq 1.$$

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

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

4 On the Regularity Conjecture

Clearly, any regular set of numbers can be generated with rules (e) only, simulating each rule $pa \rightarrow q, p \rightarrow q_h$ of a finite automaton accepting a one-letter language (for which, without restricting generality we require that every state except q_h is non-final and has at least one outgoing transition) by rules

$[_0 p]_0 \rightarrow [_0 [_0 q]_0]_0$ and $[_0 p]_0 \rightarrow [_0 [_0 q_h]_0]_0$, respectively. Here, the skin and the elementary membrane are to be seen as additional endmarkers (as otherwise only numbers ≥ 2 could be generated). Interpreting the elementary membrane as an endmarker can be avoided by additionally using a rule of type (d), i.e., of the form $[_0 [_0 q_h]_0]_0 \rightarrow [_0 q'_h]_0$.

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

$$[_0 C]_0 \Rightarrow [_0 C']_0 \quad (1)$$

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

$$C \Rightarrow^* C' \text{ implies } \underbrace{[_0 \cdots [_0 C]_0 \cdots]_0}_n \Rightarrow^* \underbrace{[_0 \cdots [_0 C']_0 \cdots]_0}_n. \quad (2)$$

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

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

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

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

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

It is easier to confirm the conjecture for the case when membrane dissolution is only allowed for elementary membranes. Then, the power of the length P system is described by the union of the finite set N with the power of a partially blind 1-register machine, starting in a state corresponding to an object from A with \bar{n} in its only register, which does not modify the register during transitions from state a to state b for which $\langle \bar{n}, a \rangle \rightarrow \langle \bar{n}, b \rangle$, and whose increment

and decrement instructions correspond to creating and dissolving elementary membranes.

It is known, see, e.g., [7], that the number sets generated by partially blind 1-register machines are equal to $NMAT = NREG$. Finally, in this case it is immediate that the membrane structure cannot change between the last time the object is in the elementary membrane and the halting.

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

5 Weak Computational Completeness

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

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

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

$$[{}_1 \underbrace{[{}_0 \cdots [{}_0}_{n_1} a \underbrace{[{}_0 \cdots [{}_0}_{n_2}]_0 \cdots]_0}_{n_2}]_0 \cdots]_0]_1 \quad (3)$$

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

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

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

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

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

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

$$[{}_h l_1]_h \rightarrow [{}_h l'_1[{}_0]_0]_h, l'_1[{}_0]_0 \rightarrow [{}_0 l_2]_0, \quad l'_1[{}_0]_0 \rightarrow [{}_0 l_3]_0.$$

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

$$[{}_h l_1]_h \rightarrow [{}_h l_2[{}_0]_0]_h, [{}_h l_1]_h \rightarrow [{}_h l_3[{}_0]_0]_h.$$

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

$$[{}_0 l_1]_0 \rightarrow l_2,$$

$$[{}_1 l_1]_1 \rightarrow [{}_1 l'_1[{}_0]_0]_1, l'_1[{}_0]_0 \rightarrow [{}_0 l''_1]_0, \quad [{}_0 l''_1]_0 \rightarrow l_3.$$

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

$$l_1[{}_0]_0 \rightarrow [{}_0 l'_1]_0, \quad [{}_0 l'_1]_0 \rightarrow l_2,$$

$$l_1[{}_1]_1 \rightarrow [{}_1 l'_1]_1, \quad [{}_1 l'_1]_1 \rightarrow l_3[{}_1]_1.$$

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

$$[{}_h l_1]_h \rightarrow [{}_h [{}_0 l''_1]_0]_h, [{}_0 l''_1]_0 \rightarrow l_3.$$

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

6 Generating Languages

In this section we no longer require that the number of membranes labeled by 1 is bounded by a constant. This allows us to generate binary languages: the result of each halting computation is the *sequence* of labels in the obtained membrane structure. The aim is to generate all recursively enumerable binary languages

(the skin is the left delimiter; the restricted rules do not permit changing it, and it is always forbidden to dissolve the skin membrane in membrane computing).

This result is not straightforward because of the following problem: we can only detect that the object is in the skin by using membrane labels. However, the approach from Section 3 does not work, since it takes unboundedly many membranes labeled by 1 to represent binary words, and we cannot find the skin by counting them with one object. Hence, we have to use encoding. We choose the following representation: the left marker is represented by bits 11, the right marker is represented by bits 10, symbol 0 is represented by bits 00, and symbol 1 is represented by bits 01.

Suppose that we have a two-symbol $\{0, 1\}$ -Turing machine \mathcal{M} , which uses the encoding mentioned above to simulate some (arbitrarily chosen) Turing machine \mathcal{M}' with two symbols 0 and 1 and the endmarkers l and r , with the following conditions:

- the head never moves to the left of the left marker;
- the right marker cannot be overwritten by 1;
- if the right marker is overwritten, then the next instructions write the right marker and move the head to the position immediately to the left of the right marker.

Clearly, any recursively enumerable language may be generated by such a Turing machine \mathcal{M}' . Moreover, without restricting generality we may assume that the routing of \mathcal{M} , corresponding to moving the right marker by \mathcal{M}' , starts by detecting that the first bit is 1, and that distinguished states are used from this point, until the marker is written in the neighboring position.

We proceed with simulating \mathcal{M} by a length P system. The sequence of labels of membranes corresponds to the contents of the tape of \mathcal{M} between the endmarkers, including them, with the additional skin membrane labeled by 1 (this membrane will never be involved in the rules, but it prevents the object from leaving to the environment), hence, the three outermost membranes are labeled by 1.

Consider an instruction $i : (q, a, b, d, q')$ where q and q' are the old and the new state, respectively, a is rewritten by b , and d denotes the direction the head has to move (L means left, R means right). In case the head is not over the endmarker, or the endmarker is not moved, i can be simulated as follows:

$$\begin{aligned} [{}_a q]_a &\rightarrow q' [{}_b]_b \quad \text{if } d = L; \\ [{}_a q]_a &\rightarrow q_1 [{}_a]_a, \quad q_1 [{}_a]_a \rightarrow [{}_b q_2]_b, \quad q_2 [{}_c]_c \rightarrow [{}_c q_2]_c, \quad 0 \leq c \leq 1 \text{ if } d = R. \end{aligned}$$

To simulate instructions $(q, r, 0, R, q_1)$, $(q_1, 0, r, L, q')$ of \mathcal{M}' , we use the following rules:

$$\begin{aligned} [{}_1 q]_1 &\rightarrow q_1 [{}_1]_1, & q_1 [{}_1]_1 &\rightarrow [{}_0 q_2]_0, & q_2 [{}_0]_0 &\rightarrow [{}_0 q_3]_0, \\ [{}_0 q_3]_0 &\rightarrow [{}_0 [{}_1 q_4]_1]_0, & [{}_1 q_4]_1 &\rightarrow [{}_1 [{}_0 q_5]_0]_1, \\ [{}_0 q_5]_0 &\rightarrow q_6 [{}_0]_0, & [{}_1 q_6]_1 &\rightarrow q_7 [{}_1]_1, & [{}_0 q_7]_0 &\rightarrow q' [{}_0]_0. \end{aligned}$$

To simulate instructions $(q, r, 0, L, q_1)$, (q_1, a, r, L, q') , $0 \leq a \leq 1$ of \mathcal{M}' , we use the following rules:

$$\begin{aligned} [1 q]_1 &\rightarrow q_1 [1]_1, & q_1 [1]_1 &\rightarrow [0 q_2]_0, & q_2 [0]_0 &\rightarrow [0 q_3]_0, \\ [0 q_3]_0 &\rightarrow q_4, & [0 q_4]_0 &\rightarrow q_5, \\ [a q_5]_a &\rightarrow q_6 [0]_0, & 0 \leq a \leq 1, & [0 q_6]_0 &\rightarrow q_7 [1]_1, & [b q_7]_b &\rightarrow q', & 0 \leq b \leq 1. \end{aligned}$$

It only remains to remove the encoding of symbols after \mathcal{M} has reached the final state q_f . The decoding can be done from left to right, one symbol at a time, moving the left marker to the right of the decoding symbol (1100 becomes 011 and 1101 becomes 111). Furthermore, since we have restricted the scope of membrane dissolution to elementary membranes only, to perform such a rewriting shortening the label sequence by one symbol, membrane labels have to be shifted outside by one position. We may assume, without restricting generality, that \mathcal{M}' halts with the head over the last symbol. Hence, in the length P system being constructed, object q_h appears in the elementary membrane. We add the following rules:

$$\begin{aligned} [0 q_h]_0 &\rightarrow s_0, & [1 s_0]_1 &\rightarrow s_{10} [0]_0, \\ [a s_{bc}]_a &\rightarrow s_{abc} [b]_b, & [0 s_{abc}]_0 &\rightarrow s_{0a} [a]_a, & a, b, c \in \{0, 1\}, \\ [1 s_{10a}]_1 &\rightarrow t [a]_a, & t [a]_a &\rightarrow [a t']_a, & 0 \leq a \leq 1, \\ t' [0]_0 &\rightarrow [1 t'']_1, & t'' [a]_a &\rightarrow [1 t_1]_1, & 0 \leq a \leq 1, \\ t_1 [0]_0 &\rightarrow [0 t_2]_0, & t_2 [a]_a &\rightarrow [a t_1]_a, & 0 \leq a \leq 1, \\ t_1 [1]_1 &\rightarrow [1 t_3]_1, & t_3 [0]_0 &\rightarrow [0 q_h]_0, \\ [1 s_{110}]_1 &\rightarrow d_1 [1]_1, & d_1 [1]_1 &\rightarrow [0 d_2]_0, \\ d_2 [1]_1 &\rightarrow [0 d_3]_0, & d_3 [0]_0 &\rightarrow [0 d_4]_0, \\ [0 d_4]_0 &\rightarrow d_5, & [0 d_5]_0 &\rightarrow d_6, & [0 d_6]_0 &\rightarrow d_7. \end{aligned}$$

The last seven rules correspond to the final phase of the decoding. When all symbols have been moved to the left of the left marker, the latter becomes adjacent to the right marker. The procedure above already deleted one of the four membranes corresponding to the markers. Symbols d_i move across the remaining three membranes that have served as markers, delete them and halt with the sequence of membrane labels corresponding exactly to the string generated by \mathcal{M} (not including the endmarkers), prefixed by 1 for the skin (for the technical reason described above).

Since the choice of the underlying machine \mathcal{M}' simulated by \mathcal{M} was arbitrary, it immediately follows that (when the number of membranes labeled by 1 is not bounded) length P systems generate all recursively enumerable languages over two symbols.

This result can be easily generalized to generating the full *RE* family, either under a suitable encoding of symbols, or with length P systems having the corresponding set of membrane labels. However, the most interesting part was to show how binary languages were generated by length P systems with two labels only.

In a similar way, one can also speak about length P systems with input (given by the membrane structure), either accepting languages or even computing partial recursive functions on them. However, if the underlying alphabet of the input

language matches the alphabet of possible membrane labels (e.g., $\{0, 1\}$), then the input has to be given in some encoding, because it is otherwise impossible to detect where the beginning of the input is.

7 Multiple Objects

In this section we consider systems not restricted to only have one object. This makes it possible to have at most one membrane labeled by 1 at a time in any computation, and to still generate all recursively enumerable sets of vectors, even with rules (b_r) , (c_r) , (d_{er}) , and (e_e) only.

We start with the membrane structure of $m + 4$ levels, all membranes labeled by 0. Initially, the skin is empty, the traveler is in the membrane inside the skin, and the next $m + 1$ membranes contain objects marking the register boundaries:

$$[_0 [_0 t_0 [_0 b_1 \underbrace{[_0 \cdots [_0]_{R_1}}]_{R_1}}]_{R_1} \cdots [_0 b_1 \cdots [_0 b_1 \underbrace{[_0 \cdots [_0]_{R_m}}]_{R_m}}]_{R_m} \cdots \underbrace{[_0 b_1 [_0]_{m+4+\sum_{i=1}^m R_i}}]_{m+4+\sum_{i=1}^m R_i}}]_{m+4+\sum_{i=1}^m R_i} ,$$

all the values R_1, \dots, R_m initially being 0.

The traveler crosses the membrane structure, counting the number of markers it meets (t_i means the marker is in the space corresponding to register i). Meeting the marker is done as follows. The traveler enters a membrane and sets its label to 1, to see whether it contains a marker:

$$t_i[_0]_0 \rightarrow [_1 t_{i,1}]_1.$$

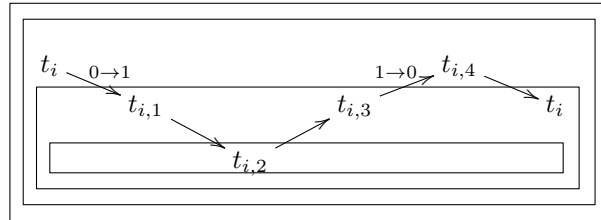
It waits for two more steps by going into the next membrane and back out:

$$t_{i,1}[_0]_0 \rightarrow [_0 t_{i,2}]_0, [_0 t_{i,2}]_0 \rightarrow t_{i,3}[_0]_0.$$

The traveler then returns to the membrane being inspected. If the label remained 1, then there was no marker there, so the traveler resets the label, exits it and continues:

$$[_1 t_{i,3}]_1 \rightarrow t_{i,4}[_0]_0, t_{i,4}[_0]_0 \rightarrow [_0 t_i]_0.$$

The following diagram illustrates the trajectory of the traveler in the absence of the marker:



If the traveler meets a marker b , the marker resets the label:

$$[_1 b_1]_1 \rightarrow b_2[_0]_0.$$

Then the traveler and the marker use the membrane label to communicate. The possible messages from the traveler to the marker are “ok, I see you”, “move one level out”, and “move one level in”. These can be communicated by the time the traveler sets the membrane label to 1 for the second time:

$$\begin{aligned} [{}_0 b_2]_0 &\rightarrow b_3[{}_0]_0, & b_3[{}_0]_0 &\rightarrow [{}_0 b_4]_0, & [{}_0 b_4]_0 &\rightarrow b_5[{}_0]_0, & b_5[{}_0]_0 &\rightarrow [{}_0 b_6]_0, \\ b_6[{}_1]_1 &\rightarrow [{}_0 b_1]_0, & b_6[{}_0]_0 &\rightarrow [{}_0 b_7]_0, & [{}_0 b_7]_0 &\rightarrow b_8[{}_0]_0, \\ [{}_0 b_8]_0 &\rightarrow b_9[{}_0]_0, & b_9[{}_0]_0 &\rightarrow [{}_0 b_{10}]_0, & [{}_0 b_{10}]_0 &\rightarrow b_{11}[{}_0]_0, & b_{11}[{}_0]_0 &\rightarrow [{}_0 b_{12}]_0, \\ b_{12}[{}_1]_1 &\rightarrow [{}_0 b_{13}]_0, & [{}_0 b_{13}]_0 &\rightarrow b_1[{}_0]_0, & b_{12}[{}_0]_0 &\rightarrow [{}_0 b'_{13}]_0, & b'_{13}[{}_0]_0 &\rightarrow [{}_0 b_1]_0. \end{aligned}$$

According to the rules described above, the marker performs the routine of exiting the current membrane (let us refer to it by h), waiting for four steps by twice exiting/re-entering the parent membrane of h , and then re-entering membrane h . When the marker b_1 exits membrane h , it resets the label to 0, letting the traveler know it is there. At the end of the routine, if the label is 1 again, then the marker does not take any further action, returning to state b_1 . If the label remained 0, the marker b_7 repeats the same routine. Depending on the label of h when it comes back as b_{12} , the marker moves inside one membrane or outside one membrane.

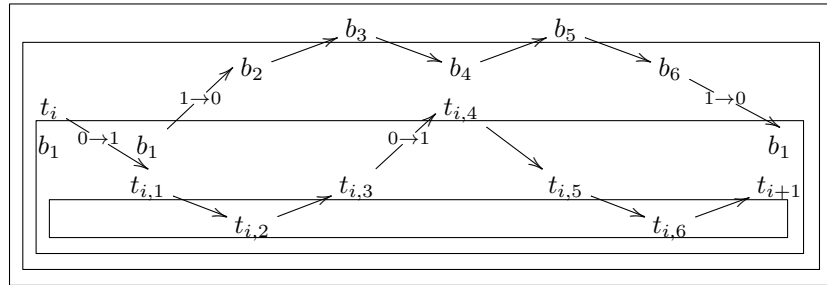
Note that the fact that evolution rules are not allowed makes it necessary to have the marker cross a membrane in order to change its state.

The rest of the construction describes the work of the traveler. To simply move inwards across the membrane structure, the following rules are used:

$$[{}_0 t_{i,3}]_0 \rightarrow t_{i,4}[{}_1]_1, t_{i,4}[{}_1]_1 \rightarrow [{}_1 t_{i,5}]_1, t_{i,5}[{}_0]_0 \rightarrow [{}_0 t_{i,6}]_0, [{}_0 t_{i,6}]_0 \rightarrow t_{i+1}[{}_1]_1.$$

The last two steps are used to wait until the marker has reset the label of h to 0 before setting the label of the child membrane of h to 1, keeping the number of membranes labeled by 1 at most 1 at any time.

The following diagram illustrates how the traveler checks the marker:



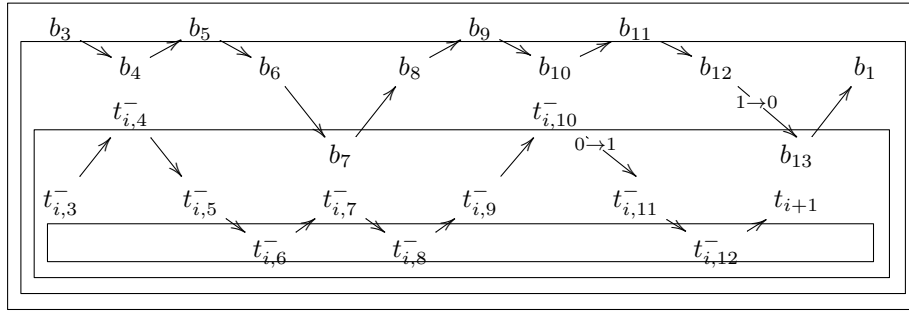
Altogether, the rules above are similar to the corresponding part of the construction in Section 3 which corresponds to traversing the membrane structure outside-in, counting the markers.

Now consider that the markers move one level inside (we add superscript $+$ to the notation of the traveler) or outside (we add superscript $-$ in that case). Such operations are required to simulate the increment and decrement instructions of

a register machine.

$$\begin{aligned}
& [{}_0 t_{i,3}^-]_0 \rightarrow t_{i,4}^- [{}_0]_0, & t_{i,4}^- [{}_0]_0 &\rightarrow [{}_0 t_{i,5}^-]_0, \\
& t_{i,5}^- [{}_0]_0 \rightarrow [{}_0 t_{i,6}^-]_0, & [{}_0 t_{i,6}^-]_0 &\rightarrow t_{i,7}^- [{}_0]_0, \\
& t_{i,7}^- [{}_0]_0 \rightarrow [{}_0 t_{i,8}^-]_0, & [{}_0 t_{i,8}^-]_0 &\rightarrow t_{i,9}^- [{}_0]_0, \\
& [{}_0 t_{i,9}^-]_0 \rightarrow t_{i,10}^- [{}_0]_0, & t_{i,10}^- [{}_0]_0 &\rightarrow [{}_1 t_{i,11}^-]_1, \\
& t_{i,11}^- [{}_0]_0 \rightarrow [{}_0 t_{i,12}^-]_0, & [{}_0 t_{i,12}^-]_0 &\rightarrow t_{i+1} [{}_0]_0.
\end{aligned}$$

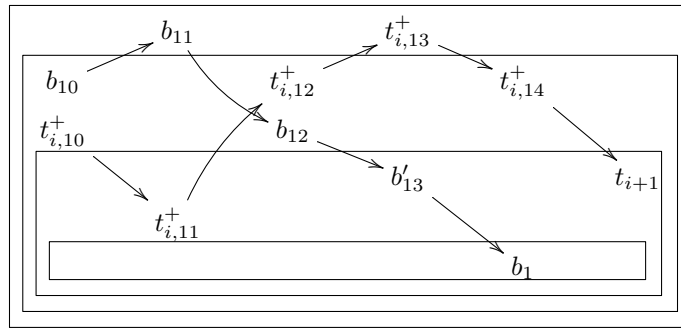
The role of the last two steps is to wait until the marker resets the label to 0 before the traveler considers the next membrane. The following diagram illustrates how the marker moves one level outside:



In the case when the marker should be moved inside, the construction is similar, except that the traveler with subscript 10 should not change the label of h to 1. Moreover, we change the path of the traveler because the marker will be entering the child membrane of h while changing its subscript from 13:

$$\begin{aligned}
& [{}_0 t_{i,3}^+]_0 \rightarrow t_{i,4}^+ [{}_0]_0, & t_{i,4}^+ [{}_0]_0 &\rightarrow [{}_0 t_{i,5}^+]_0, \\
& t_{i,5}^+ [{}_0]_0 \rightarrow [{}_0 t_{i,6}^+]_0, & [{}_0 t_{i,6}^+]_0 &\rightarrow t_{i,7}^+ [{}_0]_0, \\
& t_{i,7}^+ [{}_0]_0 \rightarrow [{}_0 t_{i,8}^+]_0, & [{}_0 t_{i,8}^+]_0 &\rightarrow t_{i,9}^+ [{}_0]_0, \\
& [{}_0 t_{i,9}^+]_0 \rightarrow t_{i,10}^+ [{}_0]_0, & t_{i,10}^+ [{}_0]_0 &\rightarrow [{}_0 t_{i,11}^+]_0, \\
& [{}_0 t_{i,11}^+]_0 \rightarrow t_{i,12}^+ [{}_0]_0, & [{}_0 t_{i,12}^+]_0 &\rightarrow t_{i,13}^+ [{}_0]_0, \\
& t_{i,13}^+ [{}_0]_0 \rightarrow [{}_0 t_{i,14}^+]_0, & t_{i,14}^+ [{}_0]_0 &\rightarrow [{}_0 t_{i+1}]_0.
\end{aligned}$$

The following diagram illustrates how the marker moves one level inside (starting with objects with subscript 10).



Traversing the structure inside-out can be done in a similar way:

$$[{}_0 r'_i]_0 \rightarrow r_i [{}_0]_0, [{}_1 r_{i,5}]_1 \rightarrow r'_{i-1} [{}_1]_1,$$

together with all rules from above operating on objects $t_i, t_{i,1}, \dots, t_{i,4}$ on the left side, accordingly renaming objects $t_i, t_{i,j}$, $1 \leq j \leq 5$ into r_i and $r_{i,j}$. Here, as opposed to the case of the outside-in travel, there is no need to wait for two last steps, because the label of h will be reset to 0 before the label of the parent of h may be changed. Clearly, traversing the structure inside-out with moving the markers may be also done in a similar fashion.

Having shown all scenarios above, we skip making the construction like that in Subsection 3, leaving the details to the interested reader.

Now consider to simulate an arbitrary register machine \mathcal{M} with $m = 3$ registers and the condition that it halts with the last two registers being empty. Simulating \mathcal{M} and finally erasing $m + 3 = 6$ innermost membranes (we view the skin as an additional endmarker, since otherwise 0 cannot be represented) implies *strong* computational completeness with at most *one* membrane labeled by 1 at a time, *only* using rules of types (b_r) , (c_r) , (d_{er}) and (e_e) . It is immediate that one can also speak about generating *PsRE* with at most *one* membrane labeled by 1 at a time, if the corresponding convention of taking the result as vectors of numbers of membranes separating the objects is used.

8 Discussion

We introduced P systems with a linear membrane structure (i.e., only one membrane is elementary) with at most one object. The result of such systems, called length P systems, is either the total number of membranes at halting, or the vector of numbers of consecutive membranes labeled by 0. In Section 3 we presented the simulation of register machines with any fixed number of registers.

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

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

We now formulate the following two *conjectures*:

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

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

In Section 6 we showed how to generate *languages* by length P systems when the number of membranes labeled by 1 is unbounded, and in Section 7 we showed how to generate vector sets with at most one membrane labeled by 1 at a time, with restricted rules, by using multiple objects.

Acknowledgements

Artiom Alhazov acknowledges project STCU-5384 *Models of high performance computations based on biological and quantum approaches* awarded by the Science and Technology Center in the Ukraine. Moreover, the authors acknowledge the useful comments of the anonymous referees.

References

1. A. Alhazov: P Systems without Multiplicities of Symbol-Objects. *Information Processing Letters* **100** (3), 2006, 124–129.
2. Alhazov, A., Freund, R., Ivanov, S: Length P Systems with a Lone Traveler. In: L. F. Macías-Ramos, M. A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, 2014, 37–46.
3. A. Alhazov, R. Freund, A. Riscos-Núñez: Membrane Division, Restricted Membrane Creation and Object Complexity in P Systems. *International Journal of Computer Mathematics* **83** (7), 2006, 529–548.
4. F. Bernardini, M. Gheorghe: Languages Generated by P Systems with Active Membranes. *New Generation Computing* **22** (4), 2004, 311–329.
5. R. Freund: Special Variants of P Systems Inducing an Infinite Hierarchy with Respect to the Number of Membranes. *Bulletin of the EATCS* **75**, 2001, 209–219.
6. A. Alhazov, M. Margenstern, S. Verlan: Fast Synchronization in P Systems. In: D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa: *Membrane Computing - 9th International Workshop*, WMC 2008, Edinburgh, Revised Selected and Invited Papers, Lecture Notes in Computer Science 5391, Springer, 2009, 118–128.
7. R. Freund, O.H. Ibarra, Gh. Păun, H.C. Yen: Matrix Languages, Register Machines, Vector Addition Systems. *Proceedings of the Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–167.
8. M. L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
9. Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
10. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
11. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*, 3 volumes. Springer, 1997.
12. The P Systems Website: <http://ppage.psystems.eu/>.

Promoters and Inhibitors in Purely Catalytic P Systems

Artiom Alhazov¹, Rudolf Freund², Sergey Verlan^{1,3}

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

² Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
E-mail: rudi@emcc.at

³ LACL, Département Informatique, Université Paris Est
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@u-pec.fr

Abstract. We consider purely catalytic P systems with two catalysts together with promoters and inhibitors on the rules. We show that computational completeness can be achieved in a deterministic way by using atomic promoters or sets of atomic inhibitors. Using atomic inhibitors computational completeness is achieved using a non-deterministic construction.

1 Introduction

Catalytic P systems are one of the first models in the area of P systems. This model allows for context-free (non-cooperative) rewriting rules like $a \rightarrow u$, where a is a single symbol, as well as contextual (cooperative) catalytic rules of the form $ca \rightarrow cu$, where c is a special object – the catalyst – present in one copy. The number of catalysts is a natural descriptive complexity parameter for such systems. It is known that systems with two catalysts are computationally complete, see [5, 14], while systems with no catalysts generate *PsREG*. The computational power of systems with one catalyst is still an open question. When additional ingredients are used, then such systems become computationally complete, e.g. when using promoters/inhibitors [11, 17] or different rule control mechanisms [8].

In purely catalytic P systems only catalytic rules are allowed, which limits the degree of the parallelism to the number of catalysts in the system. Such systems are very similar to catalytic P systems, however, not always the proofs can be translated from one model to another one because they should take into account the limitation of the parallelism. In many cases, the same results hold for purely catalytic P systems with n catalysts and catalytic P systems with $n - 1$ catalysts. For example, three catalysts are sufficient for the computational completeness of purely catalytic P systems. With one catalyst such systems are identical to sequential context-free multiset rewriting grammars, so they can

generate exactly *PsREG*. As in the previous case, the computational power of such systems with two catalysts is not yet known and different extensions have been studied in order to increase the computational power.

Promoters/inhibitors are special variants of permitting/forbidding contexts, which correspond to a single permitting/forbidding set and an empty forbidding/permitting set. It is known that non-cooperative P systems with either promoters or inhibitors of weight 2 are computationally complete, see [3]. Recently, in [1] it was shown that computational completeness cannot be achieved by deterministic non-cooperative systems with promoters, inhibitors and priorities (in the maximally parallel or the asynchronous mode, unlike the sequential mode). In [2] some interesting questions on the power of P systems in the maximally parallel mode using only priorities or restricting the weight of the promoting/inhibiting multisets were addressed.

In this paper we consider purely catalytic P systems having two catalysts and rules with promoters and inhibitors of weight 1 (consisting of a single object, and we call them atomic). We show that using atomic promoters or sets of atomic inhibitors it is possible to achieve computational completeness using a deterministic construction. In the case of atomic inhibitors our construction for computational completeness is non-deterministic. Moreover, our results can easily be adapted to the case of catalytic P systems yielding simpler proofs for the results established in [11]. We remark that the converse is not true, as the proofs from the cited article make use of a massive unbounded parallelism, so they cannot be adapted to the purely catalytic case.

2 Definitions

An *alphabet* is a finite non-empty set V of abstract *symbols*. The free monoid generated by V under the operation of concatenation is denoted by V^* ; the *empty string* is denoted by λ , and $V^* \setminus \{\lambda\}$ is denoted by V^+ . The set of non-negative integers is denoted by \mathbb{N} . The family of all finite sets of non-negative integers is denoted by *NFIN*. The family of all recursively enumerable sets of non-negative integers is denoted by *NRE*. In the following, we will use \subseteq both for the subset as well as the submultiset relation and we will represent multisets by a string notation. If we consider vectors of non-negative integer, N is replaced by Ps in these notations. The corresponding functions and relations on these sets are indicated by the prefix *Fun* and *Rel*, respectively.

2.1 Register machines.

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

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

Increases the value of register j by one, followed by a non-deterministic jump to instruction l_2 or l_3 . This instruction is usually called *increment*.

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

A *configuration* of a register machine $(l_i; v_1, \dots, v_m)$ is described by the value of the current label indicating the next instruction to be executed and the contents of each register. Computations start by executing the instruction l_0 of P , and terminate with reaching the HALT-instruction l_h .

2.2 (Purely) Catalytic P Systems

Since flattening [7] the membrane structure of a membrane system preserves both determinism and the model, in the following we restrict ourselves to consider membrane systems with only one membrane, eventually communicating with the environment, hence, in the description we omit the trivial membrane structure only consisting of the skin membrane.

Definition 1. A catalytic P system is a tuple

$$\Pi = (O, C, \Sigma, T, w, R', m_{in}, m_{out})$$

where O is a finite alphabet, $C \subset O$ is the set of catalysts, $\Sigma \subseteq O$ is the input sub-alphabet, $T \subseteq O$ is the output sub-alphabet, $w \in O^*$ is a string representing the initial multiset in the skin membrane, m_{in} and m_{out} are the input and output membranes with 1 indicating the skin membrane and 0 indicating the environment, and R' is a set of rules of one of the following forms:

- $r : a \rightarrow v$, $a \in O$, $v \in O^*$ (*non-cooperative rule*);
- $r : ca \rightarrow cv$, $a \in O$, $c \in C$, $v \in O^*$ (*catalytic rule*);
for communication with the environment, in both non-cooperative and catalytic rules, an object a in v may be replaced by $(a, come)$ meaning that the symbol a is taken from the environment, or by (a, out) meaning that the symbol a is sent out to the environment.
If only catalytic rules occur in R' , then the system is called purely catalytic.

Below, we briefly describe the semantics of catalytic P systems; more formal details can be found in [9, 14].

A configuration of the system Π is represented by a multiset of objects from O contained in the region, the set of all configurations over O is denoted by $\mathbb{C}(O)$. A rule $r : u \rightarrow v$ is applicable if the current configuration contains the multiset specified by u .

Furthermore, applicability may be controlled by *context conditions*, specified by two sets of multisets.

Definition 2. Let P and Q be (finite) sets of multisets over O . A (simple) rule with context conditions $(r; P, Q)$ is applicable to a configuration C if r is applicable, and its applicability is defined by the applicability of any of these simple rules with context conditions.

- for all $p \in P$, $p \subseteq C$, and
- for all $q \in Q$, $q \not\subseteq C$.

The more general variant of a rule with context conditions is of the form $(r; (P_1, Q_1), \dots, (P_m, Q_m))$ where any of the corresponding triples $(r; (P_i, Q_i))$, $1 \leq i \leq m$, is a simple rule with context conditions, and its applicability is defined as the applicability of any of these simple rules with context conditions.

In the definition of a (simple) rule with context conditions $(r; P, Q)$ as given above the set P is called the *permitting set*, while the set Q is called the *forbidding set* (for r). In words, context conditions are satisfied if all multisets in the permitting set are submultisets of the current configuration, and no multiset in the forbidding set is a submultiset of the current configuration.

By a promoter/inhibitor we understand an element of a permitting/forbidding set. Traditionally, P systems with promoters use the notation $r|_{p_1, \dots, p_n}$, which is equivalent to $(r; \{p_1, \dots, p_n\}; \emptyset)$. In the case of P systems with inhibitors the notation $r|_{-q_1, \dots, -q_n}$ is used, which is equivalent to $(r; \emptyset, \{q_1, \dots, q_n\})$. Finally, promoters and inhibitors consisting of one symbol are called *atomic*.

Definition 3. A P system with context conditions and priorities on the rules is a construct

$$\Pi = (O, C, \Sigma, T, w, R', R, l_{in}, l_{out}, >)$$

where $(O, C, \Sigma, T, w, R', m_{in}, m_{out})$ is a catalytic P system as defined above, R is a set of rules with context conditions and $>$ is a priority relation on the rules in R .

During a computational step the applicability of a rule can additionally be restricted by a priority relation $>$: if rule r' has priority over rule r , denoted by $r' > r$, then r cannot be applied if r' is applicable.

The computation of the system follows the usual maximally parallel derivation mode, meaning that a maximal applicable multiset of rules is chosen to be applied in each step. Also other strategies for a computational step exist, e.g., the sequential or the asynchronous derivation mode; we refer to [9, 14] for a detailed discussion on this topic. The computation step between two configurations C and C' is denoted by $C \Longrightarrow C'$, thus yielding the binary relation $\Rightarrow: \mathbb{C}(O) \times \mathbb{C}(O)$. A computation halts when there are no rules applicable to the current configuration in the corresponding mode; such a configuration is called a *halting configuration*.

In the *generative case*, a halting computation has associated a result, in the form of the number of objects present in membrane m_{out} in a halting configuration (m_{in} can be omitted). The set of non-negative integers and the set of

(Parikh) vectors of non-negative integers obtained as results of halting computations in Π are denoted by $N_{gen}(\Pi)$ and $Ps_{gen}(\Pi)$, respectively.

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

For the input being taken from the environment, i.e., for $m_{in} = 0$, we need the additional target indication *come*; $(a, come)$ means that the object a is taken into the skin from the environment (all objects there are assumed to be available in an unbounded number). The multiset of all objects taken from the environment during a halting computation then is the multiset accepted by this accepting P system, which in this case we shall call a *P automaton*, e.g., see [4]; the set of non-negative integers and the set of (Parikh) vectors of non-negative integers accepted by halting computations in Π are denoted by $N_{aut}(\Pi)$ and $Ps_{aut}(\Pi)$, respectively.

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

The family of sets $Y_{\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, computed by (purely) catalytic P systems with context conditions and priorities on the rules having at most m catalysts is denoted by $Y_{\delta}OP(cat_m, (pro_{k,l}, inh_{k',l'})_d, pri)$ ($Y_{\delta}OP(pcat_m, (pro_{k,l}, inh_{k',l'})_d, pri)$); d denotes the maximal number m in the rules with context conditions $(r, (P_1, Q_1), \dots, (P_m, Q_m))$; k and k' denote the maximum number of promoters/inhibitors in the P_i and Q_i , respectively; l and l' indicate the maximum of weights of promoters and inhibitors, respectively. If any of these numbers k , k' , l , l' is not bounded, we replace it by $*$. For $Z \in \{Fun, Rel\}$, $Y \in \{N, Ps\}$, $\delta \in \{acc, aut\}$, the corresponding families of functions/relations are denoted by $FY_{\delta}OP(cat_m, (pro_{k,l}, inh_{k',l'})_d, pri)$ and $FY_{\delta}OP(pcat_m, (pro_{k,l}, inh_{k',l'})_d, pri)$, respectively.

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

In the literature, we find a lot of restricted variants of P systems with context conditions and priorities on the rules, e.g., we may omit the priorities or the context conditions completely.

Remark 1. As in a P system $(O, C, \Sigma, T, w, R', R, m_{in}, m_{out}, >)$ the set of rules R' can easily be deduced from the set of rules with context conditions R , we omit R' in the description of the P system. Moreover, for systems having only rules

with a simple context condition, we omit d in the description of the families, and each control mechanism not used can be omitted, too.

Remark 2. It is worth to note, see also [6], that if no other control is used, the priorities can be mapped to sets of atomic inhibitors. Indeed, a rule is inhibited precisely by the left side of each rule with higher priority. This is straightforward in the case when the priority relation is assumed to be a partial order. Therefore, we will restrict ourselves to consider systems without priorities in the following.

3 Results

In this section we present the main results of the paper. First we show that purely catalytic P systems with atomic promoters are computationally complete.

Theorem 1. *The computation of a register machine can be simulated by a purely catalytic P system with only two catalysts and atomic promoters. Moreover, the simulation preserves determinism.*

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. We construct a purely catalytic P system with promoters

$$\begin{aligned} \Pi &= (O, C, \Sigma, T, w, R, m_{in}, m_{out}) \text{ where} \\ O &= C \cup \{a_r \mid 1 \leq r \leq m\} \cup \{X, Y\} \cup \{l_i \mid l_i \in B\} \cup \\ &\quad \{l'_i, l''_i, l'''_i, b_i \mid l_i : (SUB(r), l_j, l_k) \in P\}, \\ C &= \{c_1, c_2\}, \\ \Sigma, T &\subseteq \{a_r \mid 1 \leq r \leq m\}, \\ w &= l_0, \quad m_{in} = m_{out} = 1, \\ R &= \{c_1 l_i \rightarrow c_1 l_j a_r, \quad c_1 l_i \rightarrow c_1 l_k a_r \mid l_i : (ADD(r), l_j, l_k) \in P\} \\ &\quad \cup \bigcup_{l_i : (SUB(r), l_j, l_k) \in P} R_i, \end{aligned}$$

where R_i is the set of rules corresponding to the conditional decrement instruction labeled by i , consisting of the rules below.

$$\begin{array}{ll} i.1.1 : c_2 l_i \rightarrow c_2 l'_i X & i.2.1 : c_1 a_r \rightarrow c_1 b_i | l_i \\ i.1.2 : c_2 l'_i \rightarrow c_2 l''_i | b_i & i.2.2 : c_1 X \rightarrow c_1 Y \\ i.1.3 : c_2 l''_i \rightarrow c_2 l'''_i & i.2.3 : c_1 Y \rightarrow c_1 \\ i.1.4 : c_2 l'''_i \rightarrow c_2 l_j & i.2.4 : c_1 b_i \rightarrow c_1 | l'''_i \\ i.1.5 : c_2 l'_i \rightarrow c_2 l_k | Y & \end{array}$$

The simulation of the instructions of the register machine M is performed as described in the following; each configuration $(l_i; v_1, \dots, v_m)$ of M is encoded as $l_i a_1^{v_1} \dots a_m^{v_m}$ in Π .

The simulation of the increment instruction is done directly by changing the label l_i and incrementing the corresponding register. The catalyst c_1 is used for this operation.

The conditional decrement instruction $l_i : (SUB(r), l_j, l_k)$ is simulated as follows: In the first step, the rule $i.1.1$ is applicable, and if register r is not empty, then the rule $i.2.1$ is applicable, too, yielding the configuration $l'_i X a_1^{v_1} \dots a_r^{v_r-1} b_i \dots a_m^{v_m}$. Then only the rules $i.1.2$ and $i.2.2$ are applicable, yielding the configuration $l''_i Y a_1^{v_1} \dots a_r^{v_r-1} b_i \dots a_m^{v_m}$. After that only the sequence of rules $i.1.3$, $i.1.4$ and $i.2.3$, $i.2.4$ can be applied, finally yielding the configuration $l_j a_1^{v_1} \dots a_r^{v_r-1} \dots a_m^{v_m}$ corresponding to the configuration $(l_j; v_1, \dots, v_r - 1, \dots, v_m)$ of M . If the register r is empty, then in the second step only rule $i.2.2$ is applicable. In the next step, only the rules $i.1.5$ and $i.2.3$ can be applied yielding the configuration $l_k a_1^{v_1} \dots a_r^0 \dots a_m^{v_m}$, which correctly encodes the corresponding configuration of M . \square

Remark 3. We remark that only the rule with a_r on the left side actually requires the catalyst; removing the catalysts from all other rules would still yield an equivalent system, except that this system would not be purely catalytic. Hence, the catalysts could be assigned in an arbitrary way, provided no catalyst is assigned to both rules in any line.

Remark 4. We observe that is Π deterministic if M is deterministic, but also that the construction was made in such a way that the rules with the same left side and the same right side never have different promoters (without fulfilling this condition, a simpler deterministic construction could be obtained).

The next theorem shows a similar result with inhibitors. However, this proof does not preserve the determinism.

Theorem 2. *The computation of a register machine can be simulated by a purely catalytic P system with only two catalysts and atomic inhibitors.*

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. We construct a purely catalytic P system with inhibitors

$$\begin{aligned}
\Pi &= (O, C, \Sigma, T, w, R, m_{in}, m_{out}) \text{ where} \\
O &= C \cup \{l_i \mid l_i \in B\} \cup \{a_r \mid 1 \leq r \leq m\} \cup \\
&\quad \{\#, Z, h\} \cup \{l'_i, \bar{l}_i, Z_i, \bar{Z}_i, \bar{Z}'_i, b_i, \bar{b}_i \mid l_i : (SUB(r), l_j, l_k) \in P\}, \\
C &= \{c_1, c_2\}, \\
\Sigma, T &\subseteq \{a_r \mid 1 \leq r \leq m\}, \\
w &= l_0 Z, \quad m_{in} = m_{out} = 1, \\
R &= \{c_1 l_i \rightarrow c_1 l_j a_r, \quad c_1 l_i \rightarrow c_1 l_k a_r \mid l_i : (ADD(r), l_j, l_k) \in P\} \\
&\quad \cup \{c_1 \# \rightarrow c_1 \#, \quad c_2 \# \rightarrow c_2 \#, \quad c_1 l_h \rightarrow c_1 h S_1 S_2\} \\
&\quad \cup \bigcup_{l_i : (SUB(r), l_j, l_k) \in P} R_i,
\end{aligned}$$

where R_i is the set of rules corresponding to the conditional decrement instruction labeled by i , consisting of the rules below.

$$\begin{array}{lll}
i.1.1 : c_2 l_i \rightarrow c_2 \bar{l}_i S_1 S_1 S_2 \mid_{\neg S_1} & i.2.1 : c_1 a_r \rightarrow c_1 b_i \mid_{\neg S_1} & i.3.1 : c_1 Z \rightarrow c_1 Z_i \mid_{\neg S_1} \\
i.1.2 : c_2 l_i \rightarrow c_2 l'_i S_1 S_1 S_1 S_2 \mid_{\neg S_1} & i.2.2 : c_1 b_i \rightarrow \# \mid_{\neg \bar{l}_i} & i.3.2 : c_1 Z_i \rightarrow c_1 \# \mid_{\neg l'_i} \\
i.1.3 : c_2 S_1 \rightarrow c_2 \mid_{\neg S_2} & i.2.3 : c_1 b_i \rightarrow c_1 \bar{b}_i \mid_{\neg S_2} & i.3.3 : c_1 Z_i \rightarrow c_1 \bar{Z}_i \mid_{\neg S_2} \\
i.1.4 : c_2 S_2 \rightarrow c_2 \mid_{\neg h} & i.2.4 : c_1 \bar{b}_i \rightarrow c_1 & i.3.4 : c_1 \bar{Z}_i \rightarrow c_1 \# \\
i.1.5 : c_2 \bar{l}_i \rightarrow c_2 l_j \mid_{\neg S_1} & & i.3.5 : c_1 \bar{Z}_i \rightarrow c_1 \bar{Z}'_i \mid_{\neg a_r} \\
i.1.6 : c_2 l'_i \rightarrow c_2 l_k \mid_{\neg S_1} & & i.3.6 : c_1 \bar{Z}'_i \rightarrow c_1 Z
\end{array}$$

The simulation of the instructions of the register machine M is performed as described in the following; each configuration $(l_i; v_1, \dots, v_m)$ of M is encoded as $l_i Z a_1^{v_1} \dots a_m^{v_m}$ in Π .

The simulation of the increment instruction $l_i : (ADD(r), l_j, l_k) \in P$ is done directly by changing the label l_i and incrementing the corresponding register. The catalyst c_1 is used for this operation.

The conditional decrement instruction $l_i : (SUB(r), l_j, l_k)$ is simulated as follows: As can easily be seen, the rules from R_i have the property that if $c_1 x \rightarrow y \in R_i$ and $c_2 z \rightarrow u \in R_i$, then $x \neq z$. Hence, the contents of a configuration can be split into two parts C_1 and C_2 with respect to this property (objects that involve catalyst c_1 will be part of C_1 , those that involve c_2 will be part of C_2). For the case of the configuration $l_i Z a_1^{v_1} \dots a_m^{v_m}$ in Π , $C_2 = \{l_i\}$ and $C_1 = \{Z a_1^{v_1} \dots a_m^{v_m}\}$. Moreover, since the forbidding contexts of the rules $i.1.*$ do not involve objects from C_1 , their application is independent from rules involving catalyst c_1 .

First we now examine the evolution of objects from C_2 (l_j). Using catalyst c_2 , a non-deterministic guess is made about the contents of register r . If rule $i.1.1$ is used, then it is guessed that register r is not zero, yielding $\bar{l}_i S_1 S_1 S_2$, while if rule $i.1.2$ is used, then it is guessed that register r is empty, yielding $l'_i S_1 S_1 S_1 S_2$. In the first case only rule $i.1.4$ is applicable in the next step, leading to the removal of S_2 . During the next two steps only $i.1.3$ is applicable, removing the two occurrences of S_1 . Finally, symbol l_j replaces \bar{l}_i .

When using the second group of rules, the computation is one step longer (because of the presence of three copies of S_1) and yields l_k .

Now consider the evolution of objects from C_1 ($Z a_1^{v_1} \dots a_m^{v_m}$). Initially a guess is made about which is the current decrement instruction and whether using c_2 it was guessed that the corresponding register is empty or not. In some sense this permits to synchronize with the guess done in parallel using catalyst c_2 . This materializes in two groups of rules $i.2.*$ and $i.3.*$; the first one corresponds to the case when the parallel guess is that register r is not empty, while the second case corresponds to the fact that register r is empty.

Let us consider the evolution of the system in the first case. Rule $i.2.1$ takes symbol a_r corresponding to a non-empty register r and transforms it to b_i . In the next step, rule $i.2.3$ is not applicable because symbol S_2 is still present in the

configuration (it disappears on the next step as shown above). Now, if symbol \bar{l}_i is present in the configuration (this means that the parallel guess was to decrement register r being in state l_i), then no rule using catalyst c_1 is applicable. Otherwise, rule $i.2.2$ is applied introducing the trap symbol $\#$ into the system. So, if at the beginning of the third step no symbol $\#$ has been introduced, then the indices i of symbols \bar{l}_i and b_i must coincide. After that, symbol b_i is transformed to \bar{b}_i , which is erased in the next step. We note that at the same moment rule $i.1.5$ is applied, hence, the resulting configuration is $l_j Z a_1^{v_1} \dots a_r^{v_r-1} \dots a_m^{v_m}$, corresponding to the configuration $(l_j; v_1, \dots, v_r - 1, \dots, v_m)$ of M .

In the case of the application of rule $i.3.1$, a similar mechanism involving rules $i.3.2$ and $i.3.3$ permits to check if the parallel guess was that register r is empty. In this case, the rules $i.3.4$ and $i.3.5$ verify that this register is indeed empty by introducing a trap symbol if this is not the case. Finally, rule $i.3.6$ introduces back the additional symbol Z used for the zero check.

When the final label l_h is reached, rule $c_1 l_h \rightarrow h S_1 S_2$ is applied and after that the system halts as S_1 and S_2 cannot be removed, so no rule involving the first or the second catalyst can be applied. \square

We remark that the proof given above is highly non-deterministic. We conjecture that in the deterministic case atomic inhibitors should not suffice to obtain computational completeness.

The next theorem shows that forbidding conditions using sets of atomic inhibitors allow a computational completeness proof preserving the determinism.

Theorem 3. *The computation of a register machine can be simulated by a purely catalytic P system with only two catalysts and sets of atomic inhibitors. Moreover, the simulation preserves determinism.*

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. Let B_{sub} be the subset of B corresponding to the labels of SUB-instructions of M . We construct a purely catalytic P system with inhibitors

$$\begin{aligned} \Pi &= (O, C, \Sigma, T, w, R, m_{in}, m_{out}) \text{ where} \\ O &= C \cup \{X\} \cup \{m_i \mid m_i \in B\} \cup \{a_r \mid 1 \leq r \leq m\} \cup \{Z\} \\ &\quad \cup \{l'_i \mid l_i : (SUB(r), l_j, l_k) \in P\}, \\ C &= \{c_1, c_2\}, \\ \Sigma, T &\subseteq \{a_r \mid 1 \leq r \leq m\}, \\ w &= l_0, \quad m_{in} = m_{out} = 1, \\ R &= \{c_1 l_i \rightarrow c_1 l_j a_r, \quad c_1 l_i \rightarrow c_1 l_k a_r \mid l_i : (ADD(r), l_j, l_k) \in P\} \\ &\quad \cup \bigcup_{l_i : (SUB(r), l_j, l_k) \in P} R_i, \end{aligned}$$

where R_i is the set of rules corresponding to the conditional decrement instruction labeled by i , consisting of the rules below.

$$\begin{array}{ll}
i.1.1 : c_2 l_i \rightarrow c_2 l'_i X & \\
i.1.2 : (c_2 l'_i \rightarrow c_2 l_k; \emptyset, \{a_r\}) & i.2.1 : c_1 X \rightarrow c_1 \\
i.1.3 : (c_2 l'_i \rightarrow c_2 l_j; \emptyset, \{X\}) & i.2.2 : (c_1 a_r \rightarrow c_1; \emptyset, \{X\} \cup B \cup (B'_{sub} \setminus \{l'_i\}))
\end{array}$$

The simulation of the instructions of the register machine M is performed as described in the following; each configuration $(l_i; v_1, \dots, v_m)$ of M is encoded as $l_i a_1^{v_1} \dots a_m^{v_m}$ in Π .

As in the previous proofs, the simulation of the increment instruction $l_i : (ADD(r), l_j, l_k) \in P$ is done directly by changing the label l_i and incrementing the corresponding register. The catalyst c_1 is used for this operation.

The conditional decrement instruction $l_i : (SUB(r), l_j, l_k)$ is simulated as follows: In the first step the only applicable rule is $i.1.1$, which rewrites symbol l_i to $l'_i X$. In the next step rule $i.2.1$ is applicable, and only if register r is empty, then rule $i.1.2$ is applicable as well. In this way, the zero check is simulated. If register r is not empty, then in the next step rules $i.1.3$ and $i.2.2$ have to be applied performing the decrement operation. We remark that the conditions from rule $i.2.2$ allow its application only in the case when l'_i is present and X is absent. \square

Summing up the results from the preceding three theorems we obtain the following:

Theorem 4. For $Z \in \{Fun, Rel\}$, $Y \in \{N, Ps\}$, $\delta \in \{acc, aut\}$,

- $ZY_\delta OP_1(pcat_2, pro_{1,1}) = ZYRE$.
- $ZY_\delta OP_1(pcat_2, inh_{1,1}) = ZYRE$.
- $FunY_{detacc} OP_1(pcat_2, pro_{1,1}) = FunYRE$.
- $FunY_{detacc} OP_1(pcat_2, inh_{*,1}) = FunYRE$.

Generation and acceptance of a set of (vectors of) natural numbers L can be interpreted as a specific relation $\{0\} \times L$ and a function $L \times \{0\}$, respectively; thus from the previous theorem we immediately obtain the following results for generating and accepting systems:

Corollary 1. For $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$,

- $Y_\delta OP_1(pcat_2, pro_{1,1}) = YRE$.
- $Y_\delta OP_1(pcat_2, inh_{1,1}) = YRE$.
- $Y_{detacc} OP_1(pcat_2, pro_{1,1}) = YRE$.
- $Y_{detacc} OP_1(pcat_2, inh_{*,1}) = YRE$.

4 Conclusion

We have shown computational completeness for purely catalytic P systems with only two catalysts and either atomic promoters or sets of atomic inhibitors in such a way that for accepting P systems and for systems computing functions, we even get deterministic systems. The main open question is whether computational completeness can even be achieved with atomic inhibitors instead of sets of atomic inhibitors.

Acknowledgements

Artiom Alhazov acknowledges project STCU-5384 awarded by the Science and Technology Center in the Ukraine.

References

1. A. Alhazov, R. Freund: Asynchronous and Maximally Parallel Deterministic Controlled Non-Cooperative P Systems Characterize *NFIN* and *coNFIN*. *The Tenth Brainstorming Week in Membrane Computing*, vol. 1, Sevilla, 2012, 25–34, and *Membrane Computing – 13th International Conference, CMC13, Budapest* (E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil, Eds.), *Lecture Notes in Computer Science* **7762**, 2013, 101–111.
2. A. Alhazov, R. Freund: Priorities, Promoters and Inhibitors in Deterministic Non-Cooperative P Systems. In: L. F. Macías-Ramos, M. A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, 2014, 27–36.
3. A. Alhazov, D. Sburlan: Ultimately Confluent Rewriting Systems. Parallel Multiset-Rewriting with Permitting or Forbidding Contexts. In: G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa: *Membrane Computing, 5th International Workshop, WMC 2004, Milano, Revised Selected and Invited Papers, Lecture Notes in Computer Science* **3365**, Springer, 2005, 178–189.
4. E. Csuhaj-Varjú, Gy. Vaszil: P Automata or Purely Communicating Accepting P Systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing. International Workshop, WMC-CdeA 2002 Curtea de Argeș, Romania, August 19–23, 2002. Revised Papers*. *Lecture Notes in Computer Science* **2597**, Springer, 2003, pp. 219–233.
5. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient, *Theoretical Computer Science* **330**, 2, 2005, 251–266.
6. R. Freund, M. Kogler, M. Oswald, A General Framework for Regulated Rewriting Based on the Applicability of Rules. In: J. Kelemen, A. Kelemenová, *Computation, Cooperation, and Life*, Springer, *Lecture Notes in Computer Science* **6610**, 2011, 35–53.
7. R. Freund, A. Leporati, G. Mauri, A.E. Porreca, S. Verlan, C. Zandron: Flattening in (Tissue) P Systems. In: *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers. Lecture Notes in Computer Science*, 8340, 2013, 173–188.

8. R. Freund, Gh. Păun: How to Obtain Computational Completeness in P Systems with One Catalyst. In: *Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9-11, 2013, EPTCS*, **128**, 47–61.
9. R. Freund, S. Verlan: A Formal Framework for Static (Tissue) P Systems. *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, 2007, Revised Selected and Invited Papers* (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science* **4860**, 2007, 271–284.
10. O.H. Ibarra, H.-C. Yen: Deterministic Catalytic Systems are Not Universal, *Theoretical Computer Science* **363**, 2006, 149–161.
11. M. Ionescu, D. Sburlan: On P Systems with Promoters/Inhibitors. *JUCS*, **10**(5): 581–599 (2004).
12. M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
13. Gh. Păun: *Membrane Computing. An Introduction*, Springer, 2002.
14. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
15. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
16. P systems webpage. <http://ppage.psystems.eu>
17. D. Sburlan: Further results on P systems with promoters/inhibitors. *Int. J. Found. Comput. Sci.*, **17**(1): 205–221 (2006).

A Short Note on Red-Green P Automata

Bogdan Aman¹, Erzsébet Csuhaj-Varjú², and Rudolf Freund³

¹ Institute of Computer Science, Romanian Academy
Iași, Romania
Email: bogdan.aman@gmail.com

² Faculty of Informatics, Eötvös Loránd University
Budapest, Hungary
Email: csuhaj@inf.elte.hu

³ Faculty of Informatics, Vienna University of Technology
Vienna, Austria
Email: rudi@emcc.at

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

1 Introduction

In this short note we introduce the notion of red–green automata in the area of P systems. Acceptance and recognizability of finite strings by a red–green Turing machine are defined via infinite runs of the automaton on the input string and the way how to distinguish between red and green states; via infinite runs which are allowed to change between red and green states more than once, more than the recursively enumerable sets of strings can be obtained, i.e., in that way we can “go beyond Turing”. Various possibilities how to “go beyond Turing” to be already found in the literature are discussed in [10]; most of the definitions and results for red–green Turing machines are taken from this paper. In the area of P systems, first attempts to do that can be found in [6] and [9]. Computations with infinite words by P automata have been investigated in [5].

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

In this note we only focus on the concept of red–green automata for P automata, without giving formal definitions or proofs, as we assume the reader to

know the underlying notions and concepts from formal language theory (e.g., see [8]) as well as from the area of P systems (e.g., see [7]).

We only describe the *Arithmetical Hierarchy* (for example, see [2]). The Arithmetical Hierarchy is usually developed with the universal (\forall) and existential (\exists) quantifiers restricted to the integers. Levels in the Arithmetical Hierarchy are labeled as Σ_n if they can be defined by an expression beginning with a sequence of n alternating quantifiers starting with \exists . Similar expressions that start with \forall are labeled as Π_n . Σ_0 and Π_0 are defined as having no quantifiers and are equivalent. Σ_1 and Π_1 only have the single quantifier \exists and \forall , respectively. We only need to consider alternating pairs of the quantifiers \forall and \exists because two quantifiers of the same type occurring together are equivalent to a single quantifier.

Another way of looking at the Arithmetical Hierarchy is to consider the Halting Problem for Turing machines and machines equipped with an oracle solving this problem. Then one can apply the original proof for the undecidability of the Halting Problem proof to such a machine to prove it cannot solve its own Halting Problem. Then one can give an oracle for this higher level Halting Problem and generate an even higher level problem. Thus the Arithmetical Hierarchy reflects degrees of unsolvability.

A related way to extend the hierarchy of unsolvable problems is to ask if a computer program will generate an infinite number of outputs. This property can be generalized by interpreting the output of a computer as the Gödel number of another computer. Then one can ask the question “Does a program have an infinite number of outputs an infinite subset of which, when interpreted as computer programs, have an infinite number of outputs?” This can be iterated any finite number of times to create the Arithmetical Hierarchy. In that sense, the Arithmetical Hierarchy can be described as in the following table taken from [2]:

Table 1. Arithmetical Hierarchy

<i>Level</i>	<i>Question: will the computer program</i>
$\Sigma_0 = \Pi_0$	halt in fixed time
Σ_1	ever halt
Π_1	never halt
Σ_2	have at most a finite number of outputs
Π_2	have an infinite number of outputs
Σ_3	have at most a finite number of Π_2 outputs
Π_3	have an infinite number of Π_2 outputs
Σ_n	have at most a finite number of Π_{n-1} outputs
Π_n	have an infinite number of Π_{n-1} outputs

2 Red–Green Turing Machines

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

Red–green Turing machines can be seen as a type of ω -Turing machines on finite inputs with a recognition criterion based on some property of the set(s) of states visited (in)fininitely often, in the tradition of ω -automata (see [5]), i.e., we call an infinite run of the Turing machine on input w *recognizing* if and only if

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

Remark 1. In the following, “mind change” means changing the color, i.e., changing from red to green or vice versa.

To get the reader familiar with the basic idea of red–green automata, we give a short sketch of the proofs for some well-known results (see [10]):

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

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

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

2.1 Red–Green Turing Machines – Going Beyond Turing

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

Let M' be the TM recognizing L . Then construct a red–green TM M that operates on inputs w as follows: starting in red, the machine immediately switches to green and starts simulating M' on w . If M' halts (thus recognizing w), the machine switches to red and stays in red from then onward. It follows that M precisely recognizes, in fact *accepts*, the set L . *Acceptance* means that for every word not recognized by the TM it will never make an infinite number of mind changes, i.e., it finally will end up in red.

The following result characterizes the computational power of red–green Turing machines (see [10]):

Theorem 2.

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

(ii) *Red-green Turing machines accept exactly the Π_2 sets of the Arithmetical Hierarchy.*

3 The Basic Model of P Automata

The basic model of P automata as introduced in [3] and in a similar way in [4] is based on antiport rules, i.e., on rules of the form u/v , i.e., the multiset u goes out through the membrane and v comes in instead. As it is already folklore, only one membrane is needed for obtaining computational completeness with antiport rules in the maximally parallel derivation mode; the input string is defined as the sequence of terminal symbols taken in during a halting computation (without loss of generality we may assume that at most one terminal symbol is taken in from the environment in each computation step). Restricting ourselves to P automata with only one membrane as the basic model, we define a P automaton as follows:

A *P automaton* is a construct

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

where

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

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

Let us cite from [9]:

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

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

4 Red-Green P Automata

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

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

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

A stronger condition would be to divide the set of rules into “red” and “green” and to define the set of “red” and “green” multisets as those which only consist of “red” and “green” rules, respectively. But then we would have the problem how to deal with the multisets of rules consisting of rules of both colors.

5 First Results

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

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

Theorem 4.

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

(ii) *Red–green P automata accept exactly the Π_2 sets of the Arithmetical Hierarchy.*

Proof. (Sketch) Let TM be a Turing machine and RM be a register machine simulating TM having its set of internal states Q partitioned into two subsets, Q_r (the set of red states) and Q_g (the set of green states); TM operates without halting, i.e., with infinite runs on every input string. The register machine can also color its states in red and green, but when simulating the actions of TM eventually needs a green and red variant of its states and actions in order to totally stay within the same color as TM when simulating the actions of one computation step of TM . The P automaton $\Pi = (O, T, w, R)$ can simulate the actions of RM very easily, e.g., see Chapter V in [7], without introducing trap symbols, and even in a deterministic way provided RM is deterministic. Each multiset of antiport rules from R applied in the maximally parallel way contains exactly one rule of the form qu/pv where q, p are “states” of Π representing corresponding states of RM and u, v are multisets not containing any state symbol (and the other rules do not contain any state symbol, too). Hence, a configuration can be defined to exactly have the color of the “state” currently occurring in the skin region of Π , representing the corresponding state from RM . \square

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

6 Future Research

A lot of research topics wait for being investigated for P automata “going beyond Turing”, not only the different variants of defining “red” / “green” as already discussed above. For instance, the idea of having red and green configurations should also be investigated together with models of P automata which are not computationally complete, as for example dP automata.

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

References

1. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and anti-matter in membrane systems. *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, Sevilla, February 2014, 1–26.
2. P. Budnik: *What is and what will be*. Mountain Math Software, 2006.
3. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems, in: *Membrane Computing, International Workshop, WMC-CdeA 2002*, Curtea de Argeş, Romania, August 19–23, 2002, Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science* **2597**, Springer, 2003, 219–233.
4. R. Freund, M. Oswald: A short note on analysing P systems. *Bulletin of the EATCS* **78**, 2002, 231–236.
5. R. Freund, M. Oswald, L. Staiger: ω -P Automata with Communication Rules. *Workshop on Membrane Computing, 2003*, 203–217 .
6. C.S. Calude, Gh. Păun: Bio-steps beyond Turing. *Biosystems* **77** (2004), 175–194.
7. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
8. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*, 3 volumes. Springer, 1997.
9. P. Sosik, O. Valík: On Evolutionary Lineages of Membrane Systems, in: R. Freund et al. (Eds.): *WMC 2005, Lecture Notes in Computer Science* **3850** (2006), 67–78.
10. J. van Leeuwen, J. Wiedermann: Computation as an unbounded process. *Theoretical Computer Science* **429** (2012), 202–212.

Extended Simulation and Verification Platform for Kernel P Systems

Mehmet E. Bakir¹, Florentin Ipate², Savas Konur¹, Laurentiu Mierla³, and Ionut Niculescu³

¹ Department of Computer Science, University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
{mebakir1,s.konur}@sheffield.ac.uk

² Department of Computer Science, University of Bucharest
Str. Academiei nr. 14, 010014, Bucharest, Romania
florentin.ipate@ifsoft.ro

³ Department of Computer Science, University of Pitesti
Str. Targul din Vale, nr.1, 110040 Pitesti, Arges, Romania
ionutmihainiculescu@gmail.com, laurentiu.mierla@gmail.com

Abstract. We present two extensions to the software framework developed to support *kernel P systems*: a formal verification tool based on the NUSMV model checker and a large scale simulation environment based on FLAME. The use of these two tools for modelling and analysis of biological systems is illustrated with a synthetic biology example.

1 Introduction

Membrane computing [16] is a branch of natural computing inspired by the hierarchical structure of the living cell. The central model, called *P systems*, consists of a membrane structure, the regions of which contain rewriting rules operating on multisets of objects [16]. P systems *evolve* by repeatedly applying rules, mimicking chemical reactions and transportation across membranes or cellular division or death processes, and halt when no more rules can be applied. The most recent developments in this field are reported in [17].

The origins of P systems make it highly suited as a formalism for representing biological systems, especially (multi-)cellular systems and molecular interactions taking place in different locations of living cells [6]. Different simple molecular interactions or more complex gene expressions, compartment translocation, as well as cell division and death are specified using multiset rewriting or communication rules, and compartment division or dissolution rules. In the case of *stochastic P systems*, constants are associated with rules in order to compute their probabilities and time needed to be applied, respectively, according to the Gillespie algorithm [18]. This approach is based on a Monte Carlo algorithm for stochastic simulation of molecular interactions taking place inside a single volume or across multiple compartments.

The recently introduced class of *kernel P (kP) systems* [7] integrates in a coherent and elegant manner many of the features (of different P system variants) most successfully used for modelling various applications. The kP model is supported by a modelling language, called *kP-Lingua*, capable of mapping a kernel P system specification into a machine readable representation. Furthermore, the KPWORKBENCH framework that allows simulation and formal verification of the obtained models using the model checker SPIN was presented in a recent paper [4].

In this paper, we present two new extensions to KPWORKBENCH: a formal verification tool based on the NUSMV model checker [3] and a large scale simulation environment using FLAME (Flexible Large-Scale Agent Modelling Environment) [5], a platform for agent-based modelling on parallel architectures, successfully used in various applications ranging from biology to macroeconomics. The use of these two tools for modelling and analysis of biological systems is illustrated with a synthetic biology case study, pulse generator.

The paper is structured as follows. Section 2 defines the formalisms used in the paper, stochastic and kernel P systems as well as stream X-machines and communicating stream X-machine systems, which are the basis of the FLAME platform. Section 3 presents an overview on the kP-lingua language and the simulation and model checking tools. The case study and the corresponding experiments are presented in Section 4 and 5, respectively, while conclusions are drawn in Section 6.

2 Basic definitions

2.1 Stochastic and kernel P systems

Two classes of P systems, used in this paper, will be now introduced. The first model is a **stochastic P system** with its components distributed across a lattice, called *lattice population P systems* [18, 2], which have been applied to some unconventional models e.g. the genetic Boolean gates [12, 11]. For the purpose of this paper we will consider stochastic P systems with only one compartment and the lattice will be regarded as a tissue with some communication rules defined in accordance to its structure.

Definition 1. *A stochastic P system (SP system) with one compartment is a tuple:*

$$SP = (O, M, R) \quad (1)$$

where O is a finite set of objects, called alphabet; M is the finite initial multiset of objects of the compartment, an element of O^* ; R is a set of multiset rewriting rules, of the form $r_k : x \xrightarrow{c_k} y$, where x, y are multisets of objects over O (y might be empty), representing the molecular species consumed (x) and produced (y).

We consider a finite set of labels, L , and a population of SP systems indexed by this family, SP_h , $h \in L$. A lattice, denoted by Lat , is a bi-dimensional finite array of coordinates, (a, b) , with a and b positive integer numbers. Now we can define a lattice population P system, by slightly changing the definition provided in [2].

Definition 2. *A lattice population P system (LPP system) is a tuple*

$$LPP = (Lat, (SP_h)_{h \in L}, Pos, Tr) \quad (2)$$

where Lat, SP_h and L are as above and $Pos : Lat \rightarrow \{SP_h | h \in L\}$ is a function associating to each coordinate of Lat a certain SP system from the given population of SP systems. Tr is a set of translocation rules of the form $r_k : [x]_{h_1} \xrightarrow{c_k} [x]_{h_2}$, where $h_1, h_2 \in L$; this means that the multiset x from the SP system SP_{h_1} , at a certain position in Lat , will move to any of the neighbours (east, west, south, north) in Lat that contains an SP system SP_{h_2} .

The stochastic constant c_k , that appears in both definitions above, is used by Gillespie algorithm [8] to compute the next rule to be applied in the system.

One can see the lattice as a tissue system and the SP systems as nodes of it with some communication rules defined according to the neighbours and also to what they consist of.

Another class of P systems, called **kernel P systems**, has been introduced as a unifying framework allowing to express within the same formalism many classes of P systems [7, 4].

Definition 3. *A kernel P system (kP system) of degree n is a tuple*

$$k\Pi = (O, \mu, C_1, \dots, C_n, i_0) \quad (3)$$

where O is a finite set of objects, called alphabet; μ defines the membrane structure, which is a graph, (V, E) , where V are vertices indicating compartments, and E edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a compartment of the system consisting of a compartment type from T and an initial multiset, w_i over O ; i_0 is the output compartment where the result is obtained (this will not be used in the paper).

Definition 4. T is a 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 , and an execution strategy, σ_i , defined over $\text{Lab}(R_i)$, the labels of the rules of R_i .

In this paper we will use only one execution strategy corresponding to the execution of a rule in each compartment, if possible. For this reason the execution strategy will be no longer mentioned in the further definition of the systems. The rules utilised in the paper are defined below.

Definition 5. A rewriting and communication rule, from a set of rules, R_i , $1 \leq i \leq s$, used in a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$, $1 \leq i \leq n$, has the form $x \rightarrow y \{g\}$, where $x \in O^+$ and y has the form $y = (a_1, t_1) \dots (a_h, t_h)$, $h \geq 0$, $a_j \in O$ and t_j indicates a compartment type from T – see Definition 3 – with instance compartments linked to the current compartment, C_{l_i} ; t_j might indicate the type of the current compartment, i.e., t_{l_i} – in this case it is ignored; if a link does not exist (the two compartments are not in E) then the rule is not applied; if a target, t_j , refers to a compartment type that has more than one instance connected to C_{l_i} , then one of them will be non-deterministically chosen.

The definition of a rule from R_i , $1 \leq i \leq s$, is more general than the form provided above, see [7, 4], but in this paper we only use the current form. The guards, denoted by g , are Boolean conditions and their format will be discussed latter on. The guard must be true when a rule is applied.

2.2 X-machines and communicating stream X-machine systems

We now introduce the concepts of stream X-machine and communicating stream X-machine and also discuss how these are implemented in FLAME [5]. The definitions are largely from [10].

A stream X-machine is like a finite automaton in which the transitions are labelled by (partial) functions (called processing functions) instead of mere symbols. The machine has a memory (that represents the domain of the variables of the system to be modelled) and each processing function will read an input symbol, discard it and produce an output symbol while (possibly) changing the value of the memory.

Definition 6. A Stream X-Machine (SXM for short) is a tuple $Z = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0)$, where:

- Σ and Γ are finite sets called the input alphabet and output alphabet respectively;
- Q is the finite set of states;
- M is a (possibly) infinite set called memory;
- Φ is the type of Z , a finite set of function symbols. A basic processing function $\phi : M \times \Sigma \rightarrow \Gamma \times M$ is associated with each function symbol ϕ .
- F is the (partial) next state function, $F : Q \times \Phi \rightarrow 2^Q$. As for finite automata, F is usually described by a state-transition diagram.
- I and T are the sets of initial and terminal states respectively, $I \subseteq Q, T \subseteq Q$;
- m_0 is the initial memory value, where $m_0 \in M$;
- all the above sets, i.e., $\Sigma, \Gamma, Q, M, \Phi, F, I, T$, are non-empty.

A configuration of a SXM is a tuple (m, q, s, g) , where $m \in M, q \in Q, s \in \Sigma^*, g \in \Gamma^*$. An initial configuration will have the form (m_0, q_0, s, ϵ) , where m_0 is as in Definition 6, $q_0 \in I$ is an initial state, and ϵ is the empty word. A final configuration will have the form (m, q_f, ϵ, g) , where $q_f \in T$ is a terminal state. A change of *configuration*, denoted by \vdash , $(m, q, s, g) \vdash (m', q', s', g')$, is possible if $s = \sigma s'$ with $\sigma \in \Sigma$, $g' = g\gamma$ with $\gamma \in \Gamma$ and there exists $\phi \in \Phi$ such that $q' \in F(q, \phi)$ and $\phi(m, \sigma) = (\gamma, m')$. A change of configuration is called a *transition* of a SXM. We denote by \vdash^* the reflexive and transitive closure of \vdash .

A number of communicating SXMs variants have been defined in the literature. In what follows we will be presenting the communicating SXM model as defined in [10] since this is the closest to the model used in the implementation of FLAME [5] (there are however, a few differences that will be discussed later). The model defined in [10] appears to be also the most natural of the existing models of communicating SXMs since each communicating SXM is a standard SXM as defined by Definition 6. In this model, each communicating SXM has only one (global) input stream of inputs and one (global) stream of outputs. Depending on the value of the output produced by a communicating SXM, this is placed in the global output stream or is processed by a SXM component. For a more detailed discussion about the differences between various models of communicating SXMs see [14].

The following definitions are largely from [10].

Definition 7. A Communicating Stream X-Machine System (CSXMS for short) with n components is a tuple $S_n = ((Z_i)_{1 \leq i \leq n}, E)$, where:

- $Z_i = (\Sigma_i, \Gamma_i, Q_i, M_i, \Phi_i, F_i, I_i, T_i, m_{i,0})$ is the SXM with number $i, 1 \leq i \leq n$.
- $E = (e_{ij})_{1 \leq i, j \leq n}$ is a matrix of order $n \times n$ with $e_{ij} \in \{0, 1\}$ for $1 \leq i, j \leq n, i \neq j$ and $e_{ii} = 0$ for $1 \leq i \leq n$.

A CSXMS works as follows:

- Each individual *Communicating SXM* (CSXM for short) is a SXM plus an implicit input queue (i.e., of FIFO (first-in and first-out) structure) of infinite length; the CSXM only consumes the inputs from the queue.
- An input symbol σ received from the external environment (of FIFO structure) will go to the input queue of a CSXM, say Z_j , provided that it is contained in the input alphabet of Z_j . If more than one such Z_j exist, then σ will enter the input queue of one of these in a non-deterministic fashion.
- Each pair of CSXMs, say Z_i and Z_j , have two FIFO channels for communication; each channel is designed for one direction of communication. The communication channel from Z_i to Z_j is enabled if $e_{ij} = 1$ and disabled otherwise.
- An output symbol γ produced by a CSXM, say Z_i , will pass to the input queue of another CSXM, say Z_j , providing that the communication channel from Z_i to Z_j is enabled, i.e. $e_{ij} = 1$, and it is included in the input alphabet of Z_j , i.e. $\gamma \in \Sigma_j$. If these conditions are met by more than one such Z_j , then γ will enter the input queue of one of these in a non-deterministic fashion. If no such Z_j exists, then γ will go to the output environment (of FIFO structure).
- A CSXMS will receive from the external environment a sequence of inputs $s \in \Sigma^*$ and will send to the output environment a sequence of outputs $g \in \Gamma^*$, where $\Sigma = \Sigma_1 \cup \dots \cup \Sigma_n$, $\Gamma = (\Gamma_1 \setminus In_1) \cup \dots \cup (\Gamma_n \setminus In_n)$, with $In_i = \cup_{k \in K_i} \Sigma_k$, and $K_i = \{k \mid 1 \leq k \leq n, e_{ik} = 1\}$, for $1 \leq i \leq n$.

A *configuration* of a CSXMS S_n has the form $z = (z_1, \dots, z_n, s, g)$, where:

- $z_i = (m_i, q_i, \alpha_i, \gamma_i), 1 \leq i \leq n$, where $m_i \in M_i$ is the current value of the memory of Z_i , $q_i \in Q_i$ is the current state of Z_i , $\alpha_i \in \Sigma_i^*$ is the current contents of the input queue and $\gamma_i \in \Gamma_i^*$ is the current contents of the output of Z_i ;

- s is the current value of the input sequence;
- g is the current value of the output sequence.

An *initial configuration* has the form $z_0 = (z_{1,0}, \dots, z_{n,0}, s, \epsilon)$, where $z_{i,0} = (m_{i,0}, q_{i,0}, \epsilon, \epsilon)$, with $q_{i,0} \in I_i$. A *final configuration* has the form $z_f = (z_{1,f}, \dots, z_{n,f}, \epsilon, g)$, where $z_{i,f} = (m_i, q_{i,f}, \alpha_i, \gamma_i)$, with $q_{i,f} \in T_i$.

A change of configuration happens when at least one of the X-machines changes its configuration, i.e., a processing function is applied. More formally, a change of configuration of a CSXMS S_n , denoted by \models ,

$$z = (z_1, \dots, z_n, s, g) \models z' = (z'_1, \dots, z'_n, s', g'),$$

with $z_i = (m_i, q_i, \alpha_i, \gamma_i)$ and $z'_i = (m'_i, q'_i, \alpha'_i, \gamma'_i)$, is possible if one of the following is true for some i , $1 \leq i \leq n$:

1. $(m'_i, q'_i, \alpha'_i, \gamma'_i) = (m_i, q_i, \alpha_i \sigma, \epsilon)$, with $\sigma \in \Sigma_i$; $z'_k = z_k$ for $k \neq i$; $s = \sigma s'$, $g' = g$;
2. $(m_i, q_i, \sigma \alpha_i, \gamma_i) \vdash (m'_i, q'_i, \alpha'_i, \gamma)$ with $\sigma \in \Sigma_i$, $\gamma \in (\Gamma_i \setminus In_i)$; $z'_k = z_k$ for $k \neq i$; $s' = s$, $g' = g\gamma$;
3. $(m_i, q_i, \sigma \alpha_i, \gamma_i) \vdash (m'_i, q'_i, \alpha'_i, \gamma)$ with $\sigma \in \Sigma_i \cup \{\epsilon\}$, $\gamma \in (\Gamma_i \cap \Sigma_j) \cup \{\epsilon\}$ for some $j \neq i$ such that $e_{ij} = 1$; $(m'_j, q'_j, \alpha'_j, \gamma'_j) = (m_j, q_j, \alpha_j \gamma, \epsilon)$; $z'_k = z_k$ for $k \neq i$ and $k \neq j$; $s' = s$, $g' = g$;

A change of configuration is called a *transition* of a CSXMS. We denote by \models^* the reflexive and transitive closure of \models .

The correspondence between the input sequence applied to the system and the output sequence produced gives rise to the *relation computed by the system*, f_{S_n} . More formally, $f_{S_n} : \Sigma \longleftrightarrow \Gamma$ is defined by: $s f_{S_n} g$ if there exists $z_0 = (z_{1,0}, \dots, z_{n,0}, s, \epsilon)$ and $z_f = (z_{1,f}, \dots, z_{n,f}, \epsilon, g)$ an initial and final configuration, respectively, such that $z_0 \models^* z_f$ and there is no other configuration z such that $z_f \models z$.

In [14] it is shown that for any kP system, $k\Pi$, of degree n , $k\Pi = (O, \mu, C_1, \dots, C_n, i_0)$, using only rewriting and communication rules, there is a communicating stream X-machine system, $S_{n+1} = ((Z_{i,t})_{1 \leq i \leq n}, Z_{n+1}, E')$ with $n+1$ components such that, for any multiset w computed by $k\Pi$, there is a complete sequence of transitions in S_{n+1} leading to $s(w)$, the sequence corresponding to w . The first n CSXM components simulate the behaviour of the compartment C_i and the $(n+1)$ th component Z_{n+1} helps synchronising the other n CSXMs. The matrix $E' = (e'_{i,j})_{1 \leq i, j \leq n+1}$ is defined by: $e'_{i,j} = 1$, $1 \leq i, j \leq n$, iff there is an edge between i and j in the membrane structure of $k\Pi$ and $e'_{i,n+1} = e'_{n+1,i} = 1$, $1 \leq i \leq n$ (i.e., there are connections between any of the first n CSXMs and Z_{n+1} , and vice-versa). Only one input symbol σ_0 is used; this goes into the input queue of Z_{n+1} , which, in turn, sends $[\sigma_0, i]$ to each CSXM Z_i and so initializes their computation, by processing the strings corresponding to their initial multisets. Each computation step in $k\Pi$ is reproduced by a number of transitions in S_{n+1} . Finally, when the kP system stops the computation, and the multiset w is obtained in C_{i_0} , then tS_{n+1} moves to a final state and the result is sent out as an output sequence, $s(w)$.

We now briefly discuss the implementation of CSXMSs in FLAME. Basically, there are two restrictions that the FLAME implementation places on CSXMSs: (i) the associated FA of each CSXM has no loops; and (ii) the CSXMSs receive no inputs from the environment, i.e., the inputs received are either empty inputs or outputs produced (in the previous computation step) by CSXM components of the system. As explained above, a kP system is transformed into a communicating X-machine system by constructing, for each membrane, a communicating X-machine that simulates its behaviour; an additional X-machine, used for the synchronization of the others, is also used. In FLAME, however, the additional X-machine is no longer needed since the synchronization is achieved through message passing - for more details see Section 3.1 and Appendix.

3 Tools used for kP system models

The kP system models are specified with a machine readable representation, called *kP-Lingua* [4]. A slightly modified version of an example from [4] is presented below, showing how various kP systems concepts are represented in kP-Lingua.

Example 1. A type definition in kP-Lingua.

```

type C1 {
  choice {
    > 2b : 2b -> b, a(C2) .
    b -> 2b .
  }
}
type C2 {
  choice {
    a -> a, {b, 2c}(C1) .
  }
}
m1 {2x, b} (C1) - m2 {x} (C2) .

```

Example 1 shows two compartment types, C1, C2, with corresponding instances m1, m2, respectively. The instance m1 starts with the initial multiset 2x, b and m2 with an x. The rules of C1 are selected non-deterministically, only one at a time. The first rule is executed only when its guard is true, i.e., only when the current multiset has at least three b's. This rule also sends an a to the instance of type C2 linked to it. In C2 there is only a rule which is executed only when there is an a in the compartment.

The specifications written in kP-Lingua can be simulated and formally verified using a model checker called SPIN [4]. In this paper we show two further extensions, another verification mechanism based on the NUSMV model checker [3] and a large scale simulation environment using FLAME (see website⁴). These two tools, which can be downloaded from the KPWORKBENCH web page [13], will be presented below. The specification language kP-Lingua and the verification and simulation tools are all integrated within KPWORKBENCH [4].

3.1 Simulation

The ability of simulating *kernel P systems* is one important aspect provided by a set of tools supporting this formalism. Currently, there are two different simulation approaches (a performance comparison can be found in [15]). Both receive as input a kP-Lingua model and outputs a trace of the execution, which is mainly used for checking the evolution of a system and for extracting various results out of the simulation.

KPWorkbench Simulator. KPWORKBENCH contains a simulator for kP system models and is written in the C# language. The simulator is a command line tool, providing a means for configuring the traces of execution for the given model, allowing the user to explicitly define the granularity of the output information by setting the values for a concrete set of parameters:

- *Steps* - a positive integer value for specifying the maximum number of steps the simulation will run for. If omitted, it defaults to 10

⁴ <http://flame.ac.uk>

- *SkipSteps* - a positive integer value representing the number of steps to skip the output generation for. By using this parameter, the simulation trace will be generated from the step next to the current specified one, onward. If not set, the default value is 0
- *RecordRuleSelection* - defaulting to *true*, takes a boolean value on which to decide if the rule selection mechanism defined by the execution strategy will be generated into the output trace
- *RecordTargetSelection* - if *true* (which is also the default value), traces the resolution of the communicating rules, outputting the non-deterministically selected membrane of a specified type to send the objects to
- *RecordInstanceCreation* - defaulting to *true*, specifies if the membrane creation processed should be recorded into the output simulation trace
- *RecordConfigurations* - if *true* (being also the default setting), generates as output, at the end of a step, the multiset of objects corresponding to each existing membrane
- *ConfigurationsOnly* - having precedence over the other boolean parameters, sets the value of the above flags to *false*, except the one of the *RecordConfigurations*, causing the multiset configuration for each of the existing membranes to be the only output into the simulation trace. The default value is *false*

It's worth mentioning that the KPWORKBENCH SIMULATOR tool, including the simulator, are also available for the Unix-based operating systems supported by the Mono framework⁵.

FLAME-based Simulator The agent-based modeling framework FLAME is also used for simulating kP-Lingua specifications, one of the main advantages of this approach being its big scalability degree and efficiency for simulating large scale models.

A general FLAME simulation requires the provision of a model for specifying the agents representing the definitions of communicating X-machines, whose behaviour is to be simulated, together with input data representing the initial values of the memory for the generated X-machines. The model specification is composed of an xml file defining the structure of the agents, while their behaviour is provided as a set of functions in the C programming language.

In order to be able to simulate kernel P system models using the FLAME framework, an automated model translation is made for converting the kP-Lingua specification into the above mentioned formats. Thus, the various compartments defined into the kP-Lingua model are translated into agent definitions, while the rule execution strategies corresponds to the transitions describing the behaviour of the agents. More specifically, each membrane of the kP system is represented by an agent. The rules are stored together with the membrane multiset as agent data. For each type of membrane from the kP system, a type of agent is defined, and for each execution strategy of the membrane, states are created in the X-machine. Transitions between the two states are represented by C functions that are executed in FLAME when passing from one state to another. Each type of strategy defines a specific function that applies the rules according to the execution strategy. A detailed description of the algorithm for translating a kP system into FLAME is given in Appendix.

Each step of the simulation process modifies the memory of the agents, generating at the same time output xml files representing the configuration of the corresponding membranes at the end of the steps. The granularity level of the information defining the simulation traces is adjustable by providing a set of concrete parameters for the input data set.

3.2 Model Checking

KPWORKBENCH already integrates the SPIN model checker [9]. A more detailed account can be found in [4]. In this paper, we also integrate the NUSMV model checker [3] to the KPWORKBENCH

⁵ <http://mono-project.com>

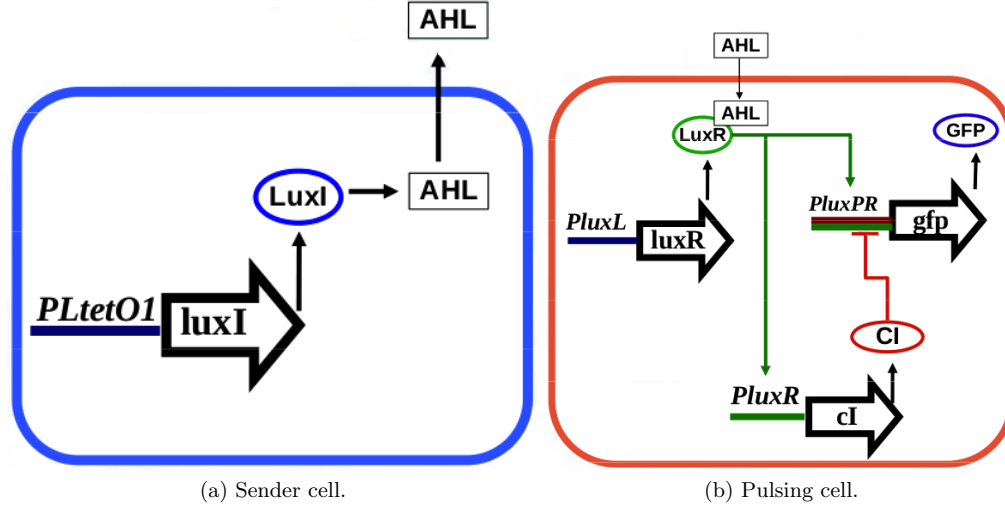


Fig. 1: Two cell types of the pulse generator system (taken from [2]).

platform to be able to verify branching-time semantics. NuSMV is designed to verify synchronous and asynchronous systems. Its high-level modelling language is based on *Finite State Machines* (FSM) and allows the description of systems in a modular and hierarchical manner. NuSMV supports the analysis of specification expressed in *Linear-time Temporal Logic* (LTL) and *Computation Tree Logic* (CTL). NuSMV employs *symbolic* methods, allowing a compact representation of the state space to increase the efficiency and performance. The tool also permits conducting simulation experiments over the provided FSM model by generating traces either interactively or randomly.

4 Case study: Pulse Generator

The *pulse generator* [1] is a synthetic biology system, which was analysed stochastically in [2, 12]. It is composed of two types of bacterial strains: *sender* and *pulsing* cells (see Figure 1). The sender cells produce a signal (30C6-HSL) and propagates it through the pulsing cells, which express the green fluorescent protein (GFP) upon sensing the signal. The excess of the signalling molecules are propagated to the neighbouring cells.

Sender cells synthesize the signalling molecule 30C6-HSL (AHL) through the enzyme LuxI, expressed under the constitutive expression of the promoter *PLtetO1*. Pulsing cells express GFP under the regulation of the *PluxPR* promoter, activated by the LuxR_30C6_2 complex. The LuxR protein is expressed under the control of the *PluxL* promoter. The GFP production is repressed by the transcription factor CI, codified under the regulation of the promoter *PluxR* that is activated upon binding of the transcription factor LuxR_30C6_2.

4.1 Stochastic model

The formal model consists of two bacterial strains, each one is represented by an SP system model. So, $L = \{sender, pulsing\}$, describes these two labels and accordingly:

$$SP_h = (O_h, M_h, R_h), h \in L \quad (4)$$

where

$$O_{sender} = \{\text{PLtet01_geneLuxI}, \text{proteinLuxI}, \text{proteinLuxI_Rib}, \text{rnaLuxI}, \\ \text{rnaLuxI_RNAP}, \text{signal30C6}\}$$

$$M_{sender} = \text{PLtet01_geneLuxI}$$

$$R_{sender} = \{r_1 : \text{PLtet01_geneLuxI} \xrightarrow{k_1} \text{PLtet01_geneLuxI} + \text{rnaLuxI_RNAP} \quad k_1 = 0.1, \\ r_2 : \text{rnaLuxI_RNAP} \xrightarrow{k_2} \text{rnaLuxI} \quad k_2 = 3.36, \\ r_3 : \text{rnaLuxI} \xrightarrow{k_3} \text{rnaLuxI} + \text{proteinLuxI_Rib} \quad k_3 = 0.0667, \\ r_4 : \text{rnaLuxI} \xrightarrow{k_4} \quad k_4 = 0.004, \\ r_5 : \text{proteinLuxI_Rib} \xrightarrow{k_5} \text{proteinLuxI} \quad k_5 = 3.78, \\ r_6 : \text{proteinLuxI} \xrightarrow{k_6} \quad k_6 = 0.067, \\ r_7 : \text{proteinLuxI} \xrightarrow{k_7} \text{proteinLuxI} + \text{signal30C6} \quad k_7 = 5\}$$

and

$$O_{pulsing} = \{\text{CI2}, \text{LuxR2}, \text{PluxL_geneLuxR}, \text{PluxPR_CI2_geneGFP}, \\ \text{PluxPR_LuxR2_CI2_geneGFP}, \text{PluxPR_LuxR2_geneGFP}, \text{PluxPR_geneGFP}, \\ \text{PluxR_LuxR2_geneCI}, \text{PluxR_geneCI}, \text{proteinCI}, \text{proteinCI_Rib}, \text{proteinGFP}, \\ \text{proteinGFP_Rib}, \text{proteinLuxR}, \text{proteinLuxR_30C6}, \text{proteinLuxR_Rib}, \text{rnaCI}, \\ \text{rnaCI_RNAP}, \text{rnaGFP}, \text{rnaGFP_RNAP}, \text{rnaLuxR}, \text{rnaLuxR_RNAP}, \text{signal30C6}\}$$

$$M_{pulsing} = \text{PluxL_geneLuxR}, \text{PluxR_geneCI}, \text{PluxPR_geneGFP}.$$

The set of rules ($R_{pulsing}$) is presented in Table 6 (Appendix).
The translocation rules are:

$$Tr = \{r_1 : [\text{signal30C6}]_{sender} \xrightarrow{k_1} [\text{signal30C6}]_{pulsing} \quad k_1 = 1.0, \\ r_2 : [\text{signal30C6}]_{sender} \xrightarrow{k_2} [\text{signal30C6}]_{sender} \quad k_2 = 1.0, \\ r_3 : [\text{signal30C6}]_{pulsing} \xrightarrow{k_3} [\text{signal30C6}]_{pulsing} \quad k_3 = 1.0\}.$$

The lattice, given by Lat , is an array with n rows and m columns of coordinates (a, b) , where $0 \leq a \leq n - 1$ and $0 \leq b \leq m - 1$. The values n and m will be specified for various experiments conducted in this paper. In all the experiments the first column will be associated with *sender* SP systems and the rest with *pulsing* systems. Formally, $Pos(a, 0) = SP_{sender}$, $0 \leq a \leq n - 1$, and $Pos(a, b) = SP_{pulsing}$, $0 \leq a \leq n - 1$ and $1 \leq b \leq m - 1$.

4.2 Nondeterministic model

Non-deterministic models are used for qualitative analysis. They are useful for detecting the existence of molecular species rather than for measuring their concentration. A typical non-deterministic model can be obtained from a stochastic model by removing the kinetics constants.

More precisely, one can define two types corresponding to the two bacterial strains in accordance with Definition 4, namely $T = \{sender, pulsing\}$, and the corresponding rule sets, R'_{sender} and $R'_{pulsing}$. The rules from R'_{sender} are obtained from R_{sender} and $r_1, r_2 \in Tr$, and those from $R'_{pulsing}$ are obtained from $R_{pulsing}$ and $r_3 \in Tr$, by removing the kinetic rates. For each rule from the set Tr , namely $r_k : [x]_{h_1} \xrightarrow{c_k} [x]_{h_2}$, the corresponding rule of the kP system will be $r_k : x \rightarrow x(t)$, where $t \in T$. The execution strategies are those described in the associated definitions of the kP systems.

The kP system with $n \times m$ components is given, in accordance with Definition 3, by the graph with vertices $C_{a,b} = (t_{a,b}, w_{a,b})$, where $t_{a,b} \in T$ and $w_{a,b}$ is the initial multiset, $0 \leq a \leq n-1$, $0 \leq b \leq m-1$; and edges where each component $C_{a,b}$, with $0 \leq a \leq n-2$, $0 \leq b \leq m-2$, is connected to its east, $C_{a,b+1}$, and south, $C_{a+1,b}$, neighbours. The types of these components are $t_{a,0} = \textit{sender}$, $0 \leq a \leq n-1$, and $t_{a,b} = \textit{pulsing}$, $0 \leq a \leq n-1$ and $1 \leq b \leq m-1$. The initial multisets are $w_{a,0} = M_{\textit{sender}}$, $0 \leq a \leq n-1$, and $w_{a,b} = M_{\textit{pulsing}}$, $0 \leq a \leq n-1$ and $1 \leq b \leq m-1$.

So, one can observe the similitude between the lattice and function *Pos* underlying the definition of the LPP system and the graph and the types associated with the kP system.

4.3 Nondeterministic simplified model

In order to relieve the state explosion problem, models can also be simplified by replacing a long chain of reactions by a simpler rule set which will capture the starting and ending parts of this chain, and hence eliminating species that do not appear in the new rule set. With this transformation we achieve a simplification of the state space, but also of the number of transitions associated with the model.

The non-deterministic system with a set of compacted rules for the sender cell is obtained from the kP system introduced above and consists of the same graph with the same types, T , and initial multisets, $w_{a,b}$, $0 \leq a \leq n-1$, $0 \leq b \leq m-1$, but with simplified rule sets obtained from $R'_{\textit{sender}}$ and $R'_{\textit{pulsing}}$, and denoted $R''_{\textit{sender}}$, defined as follows:

$$R''_{\textit{sender}} = \{r_1 : \text{PLtet01.geneLuxI} \rightarrow \text{PLtet01.geneLuxI} + \text{rnaLuxI.RNAP}, \\ r_2 : \text{proteinLuxI} \rightarrow, \\ r_3 : \text{proteinLuxI} \rightarrow \text{proteinLuxI} + \text{signal30C6}, \\ r_4 : \text{signal30C6} \rightarrow \text{signal30C6} (\textit{pulsing}), \\ r_5 : \text{signal30C6} \rightarrow \text{signal30C6} (\textit{sender})\}$$

and $R''_{\textit{pulsing}}$ is defined in Table 7 (Appendix).

5 Experiments

5.1 Simulation

The simulation tools have been used to check the temporal evolution of the system and to infer various information from the simulation results. For a kP system of 5×10 components, which comprises 25 sender cells and 25 pulsing cells, we have observed the production and transmission of the signaling molecule from the sender cells to the furthest pulsing cell and the production of the green fluorescent protein.

FLAME Results As explain earlier, in FLAME each agent is represented by an X-machine. When the X-machine reaches the final state, the data is written to the hard disk and it is then used as input for the next iteration. Since the volume of data increases with the number of membranes, the more membranes we have, the more time for reading and writing data from or to the hard disk is required. Consequently, when the number of membranes is large, the time required by the read and write operations increases substantially and so the simulation may become infeasible⁶. For example, for the pulsing generator system it was difficult to obtain simulation results after 100,000 steps; the execution time for 100,000 steps was approximately one hour.

⁶ On the other hand, the distributed architecture of FLAME allows the simulation to be run on parallel supercomputers with great performance improvements, but this is beyond the scope of this paper.

Table 1: FLAME results.

Step Interval	sender _{1,1}		pulse _{5,9}	
	signal30C6	signal30C6	proteinGFP	proteinGFP
0 – 10,000	Exist	Exist	None	None
10,001 – 20,000	Exist	Exist	None	None
20,001 – 30,000	Exist	Exist	None	None
30,001 – 40,000	Exist	Exist	None	None
40,001 – 50,000	Exist	Exist	None	None
50,001 – 60,000	Exist	Exist	None	None
60,001 – 70,000	Exist	Exist	None	None
70,001 – 80,000	Exist	Exist	None	None
80,001 – 90,000	Exist	Exist	None	None
90,001 – 99,666	Exist	Exist	None	None
99,667 – 100,000	Exist	Exist	Exist	Exist

Table 2: KPWORKBENCH SIMULATOR results.

Step Interval	sender _{1,1}		pulse _{5,10}	
	signal30C6	signal30C6	proteinGFP	proteinGFP
0 – 300,000	Exist	Exist	None	None
300,001 – 600,000	Exist	Exist	None	None
600,001 – 900,000	Exist	Exist	None	None
900,001 – 1,200,000	Exist	Exist	None	None
1,200,001 – 1,500,000	Exist	Exist	None	None
1,500,001 – 1,800,000	Exist	Exist	None	None
1,800,001 – 2,100,000	Exist	Exist	None	None
2,100,001 – 2,400,000	Exist	Exist	None	None
2,400,001 – 2,700,000	Exist	Exist	Exist	Exist
2,700,001 – 3,000,000	Exist	Exist	None	None

The signaling molecule **signal30C6** appeared for the first time in the sending cell **sender_{1,1}** at the 27th step ; after that , it appeared and disappeared many times. In the pulsing cell **pulse_{5,9}**, the signaling molecule appeared for the first time at 4963 steps, while the **proteinGFP** was produced for the first time after 99,667steps.

The results of the FLAME simulation show that the signaling molecules produced in the sending cells are propagated to the pulsating cells which, in turn, produce the **proteinGFP**. The results of the simulation are given in Table 1. In 100,000 steps (the maximum number of steps considered for the FLAME simulation), the farthest cell in which **proteinGFP** was produced was **pulse_{5,9}** - this was produced after 99,667 steps.

KPWorkbench Simulator Results KPWORKBENCH SIMULATOR is a specialised simulation tool and provides better results, in terms of execution time, then a general purpose simulation environment like FLAME. This is mainly due to the fact that this approach makes the simulation to be performed in a single memory space, that scales according to the number of membranes used in the model and the number of objects resulting from applying the rules in each simulation step.

Table 2 presents the production and availability of the signaling molecule at the first sender cell (i.e. **sender_{1,1}**) and the transmission of the signaling molecule and the production of the green florescent protein at the furthest pulsing cell (i.e. **pulse_{5,10}**).

Table 3: Property patterns used in the verification experiments.

Prop.	Informal specification and the corresponding CTL translations
1	<i>X will eventually be produced.</i> $EF X>0$
2	<i>The availability/production of X will eventually lead to the production of Y.</i> $AG (X \Rightarrow EF Y)$
3	<i>Y cannot be produced before X is produced.</i> $\neg E (X=0 \cup Y>0)$

We have run the simulator for 3,000,000 time steps. The `sender1,1` cell was able to produce the signaling molecule at 22 steps, and later produced more signaling molecules. The `pulse5,10` cell, as the furthest pulse generator cell, was able to receive the signaling molecule at the 5474 step. But, the production of `proteinGFP` was possible at the 2,476,813 step, and it remained inside the cell until the 2,476,951 step, then it was consumed.

The simulation results show that the signaling molecule can be produced and transmitted by a sender cell. In addition, a pulse generator cell can have a signaling molecule only after a sender cell sends it, and can use the signal for the production of `proteinGFP` in later steps.

5.2 Verification

The properties of the system are verified using the NuSMV model checker, fully integrated into the KPWORKBENCH platform. In this section, we first verify individual cell properties, and then verify the properties of the whole kP system, involving multiple cells that are distributed within the lattice and interact with each other via the translocation rules.

Our verification results show that when the cell population is small, the properties can be verified using reasonable computational resources. However, given that the complete rule set is used, when the number of cell increases, verification becomes no longer feasible due to the state explosion problem. To mitigate this problem, we have used simplified rule set to verify the cell interaction properties when the cell populations is large.

Complete Rule Sets. Experiments under this section are conducted on a small population of multi-cellular systems including the complete set of rules. We have analysed two pulse-generator systems that differ only in the number of pulse generator cells. The first group consists of one sender cell and one pulse generator cell, i.e. 1×2 components, whereas the second group has one more pulse generator cell, i.e. 1×3 components.

For our experiments, we use the property patterns provided in Table 3. Table 4 shows the verification results for the properties given in Table 3 using two different groups. NuSMV has returned TRUE for all the properties. In the group with 1×2 components, we have verified that the sender cell (`sender1`) can produce a signalling molecule and transmit it to the pulsing cell (`pulsing1`). In addition, the pulse generator cell can use that signal to produce the green florescent protein (`proteinGFP`). In the group with 1×3 components, we have verified similar properties. In addition, we have verified that the first pulsing cell (`pulsing1`) can transmit the signalling molecule to the second pulsing cell (`pulsing2`).

Reduced Rule Sets. Using a larger sets of components, we want to prove that the signalling molecule can be transmitted to the further pulsing cells. However, when we increase the number of cells, verification becomes no longer feasible due to the state explosion problem. In order to achieve the verification results within a reasonable time, we have compacted the rules sets such

Table 4: Verification experiments for the complete rule sets.

Lattice	Property	X, Y
1 × 2	Prop. 1	X = sender ₁ .signal30C6 X = pulsing ₁ .signal30C6
	Prop. 2	X = pulsing ₁ .signal30C6, Y = pulsing ₁ .proteinGFP X = sender ₁ .signal30C6, Y = pulsing ₁ .proteinGFP
	Prop. 3	X = pulsing ₁ .signal30C6, Y = pulsing ₁ .proteinGFP X = sender ₁ .signal30C6, Y = pulsing ₁ .proteinGFP
1 × 3	Prop. 1	X = pulsing ₂ .signal30C6 X = pulsing ₂ .proteinGFP
	Prop. 2	X = pulsing ₁ .signal30C6, Y = pulsing ₂ .proteinGFP X = sender ₁ .signal30C6, Y = pulsing ₂ .proteinGFP
	Prop. 3	X = pulsing ₁ .signal30C6, Y = pulsing ₂ .signal30C6 X = sender ₁ .signal30C6, Y = pulsing ₂ .signal30C6

Table 5: Verification experiments for the reduced rule sets.

Lattice	Property	X, Y
1 × 5	Prop. 1	X = pulsing ₄ .signal30C6 X = pulsing ₄ .proteinGFP
	Prop. 2	X = pulsing ₁ .signal30C6, Y = pulsing ₄ .proteinGFP X = sender ₁ .signal30C6, Y = pulsing ₄ .proteinGFP
	Prop. 3	X = pulsing ₃ .signal30C6, Y = pulsing ₄ .signal30C6 X = pulsing ₃ .signal30C6, Y = pulsing ₄ .proteinGFP

that an entire chain of reactions is replaced by a fewer simple rules. Consequently, the overall number of interactions is reduced and all the species which do not appear in the new set of rules are removed from the model. These changes are made in the non-deterministic models as these are used for qualitative analyses where the concentration of certain molecules is not significant or chain of reaction already analysed can be replaced by some abstractions mimicking their behaviour through simpler rewriting mechanisms.

Here, we define a group of cells with 1 × 5 components, where 1 sender and 4 pulsing cells are placed in row. For this scenario, we could verify the same properties in Table 4 using the reduced rule sets (as defined in Section 4.3). In addition, we have verified additional properties to analyse the other pulsing cells. Table 5 shows these properties, for which NuSMV has returned TRUE. The verification results show that the sender cell can produce the signalling molecule and transmit it to the adjacent pulsing cell, and the pulsing cells can use the signalling molecule to produce proteinGFP and transmit it to the its neighbour pulsing cells.

6 Conclusions

In this paper, we have presented two extensions to KPWORKBENCH: a formal verification tool based on the NuSMV model checker and a large scale simulation environment using FLAME, a platform for agent-based modelling on parallel architectures. The use of these two tools for modelling and analysis of biological systems is illustrated with a pulse generator, a synthetic biology system. We have provided both the stochastic model as SP systems and the non-deterministic model as kP systems as well as a reduced model. We have also provided both simulation and verification results, confirming the desired behaviour of the pulse generator system.

Future work will involve modeling, simulation and verification of even more complex biological systems as well as performance comparisons of simulators and model checking tools integrated within KPWORKBENCH.

Acknowledgements. The work of FI, LM and IN was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (project number: PN-II-ID-PCE-2011-3-0688). SK acknowledges the support provided for synthetic biology research by EPSRC ROADBLOCK (project number: EP/I031812/1). MB is supported by a PhD studentship provided by the Turkey Ministry of Education.

The authors would like to thank Marian Gheorghe for his valuable comments to this paper.

References

1. Basu, S., Mehreja, R., Thiberge, S., Chen, M.T., Weiss, R.: Spatio-temporal control of gene expression with pulse-generating networks. *PNAS* 101(17), 6355–6360 (2004)
2. Blakes, J., Twycross, J., Konur, S., Romero-Campero, F.J., Krasnogor, N., Gheorghe, M.: Infobiotics Workbench: A P systems based tool for systems and synthetic biology. In: [6], pp. 1–41. Springer (2014)
3. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV version 2: An open source tool for symbolic model checking. In: Proc. International Conference on Computer-Aided Verification (CAV 2002). LNCS, vol. 2404. Springer, Copenhagen, Denmark (July 2002)
4. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierlă, L.: Model checking kernel P systems. In: 14th International Conference on Membrane Computing. LNCS, vol. 8340, pp. 151–172. Springer (2013)
5. FLAME: Flexible large-scale agent modeling environment (available at <http://www.flame.ac.uk/>)
6. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing in Systems and Synthetic Biology. Springer (2014)
7. Gheorghe, M., Ipate, F., Dragomir, C., Mierlă, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P systems - Version 1. 12th BWMC pp. 97–124 (2013)
8. Gillespie, D.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics* 22(4), 403–434 (1976)
9. Holzmann, G.J.: The model checker SPIN. *IEEE Transactions on Software Engineering* 23(5), 275–295 (1997)
10. Ipate, F., Bălănescu, T., Kefalas, P., Holcombe, M., Eleftherakis, G.: A new model of communicating stream X-machine systems. *Romanian Journal of Information Science and Technology* 6, 165–184 (2003)
11. Konur, S., Gheorghe, M., Dragomir, C., Ipate, F., Krasnogor, N.: Conventional verification for unconventional computing: a genetic XOR gate example. *Fundamenta Informaticae* p. To Appear (2014)
12. Konur, S., Gheorghe, M., Dragomir, C., Mierla, L., Ipate, F., Krasnogor, N.: Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. *ACS Synthetic Biology* p. To Appear (2014)
13. KPWorkbench (available at <http://muvet.ifsoft.ro/kpworkbench/>)
14. Niculescu, I.M., Gheorghe, M., Ipate, F., Stefanescu, A.: From kernel P systems to X-machines and FLAME. *Journal of Automata, Languages and Combinatorics* To appear (2014)
15. Niculescu, I., Ipate, F., Bakir, M.E., Konur, S., Gheorghe, M.: High performance simulations of kernel P systems. In: The 16th IEEE International Conference on High Performance Computing and Communications. p. To Appear (2014)
16. Păun, G.: Computing with membranes. *Journal of Computer and System Sciences* 61(1), 108–143 (2000)
17. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
18. Romero-Campero, F.J., Twycross, J., Cao, H., Blakes, J., Krasnogor, N.: A multiscale modeling framework based on P systems. In: Membrane Computing, LNCS, vol. 5391, pp. 63–77. Springer (2009)

Appendix

Algorithm 1 Transforming a kP Systems into Flame algorithm

```

1: procedure ADDTTRANSITION(startState, stopState, strategy, guard)
  ▷ procedure adding the appropriate transition strategy to the current agent stack given as parameter
  ▷ and FLAME function applying rules conforming to execution strategy
  ▷ guard is an optional parameter that represents the transition guard
2:   if strategy is Sequence then
3:     agentTtransitions.Push(startState, stopState, SequenceFunction, guard)
  ▷ FLAME function SequenceFunction applies rules in sequentially mode
4:   else if strategy is Choice then
5:     agentTtransitions.Push(startState, stopState, ChoiceFunction, guard)
  ▷ FLAME function ChoiceFunction applies rules in choice mode
6:   else if strategy is ArbitraryParallel then
7:     agentTtransitions.Push(startState, stopState, ArbitraryParallelFunction, guard)
  ▷ FLAME function ArbitraryParallelFunction applies rules in arbitrary parallel mode
8:   else if strategy is MaximalParallel then
9:     agentTtransitions.Push(startState, stopState, MaximalParallelFunction, guard)
  ▷ FLAME function MaximalParallelFunction applies rules in maximal parallel mode
10:  end if
11: end procedure
12:
  ▷ main algorithm for traforming a kP system into Flame
13:
14: agentsStates.Clear()
15: agentsTtransitions.Clear()
  ▷ empty state and transition stacks of agents
16: foreach membrane in kPSystem do
  ▷ for each membrane of kP system build corresponding agent, consisting of states and transitions
17:   agentStates.Clear()
18:   agentTtransitions.Clear()
  ▷ empty state and transition stacks of agent that is built for the current membrane
19:   agentStates.Push(startState)
  ▷ adding the initial state of the X machine
20:   agentStates.Push(initializationState)
  ▷ adding initialization state
21:   agentTtransitions.Push(startState, initializationState)
  ▷ adding transition between the initial and initialization states; this transition performs objects
  ▷ allocation on rules and other initializations
22:   foreach strategy in membrane do
  ▷ for each strategy of the current membrane the corresponding states and transitions are built
23:     previousState = agentStates.Top()
  ▷ the last state is stored in a temporary variable
24:     if is first strategy and strategy.hasNext() then
  ▷ when the strategy is the first of several, state and transition corresponding to the execution
  ▷ strategy are added
25:       agentStates.Push(strategy.Name)
26:       AddTtransition(previousState, strategy.Name, strategy)
27:     else
28:       if not strategy.hasNext() then
  ▷ if it is the last strategy, the transition corresponding to the execution strategy is added
29:         AddTtransition(previousState, applyChangesState, strategy)
30:       else

```

Algorithm 1 Transforming a kP Systems into Flame algorithm (continued)

```

31:     agentStates.Push(strategy.Name)
    ▷ add corresponding state of the current strategy
32:     if strategy.Previous() is Sequence then
    ▷ verify that previous strategy is of sequence type
33:         AddTtransition(previousState, strategy.Name, strategy, IsApplyAllRules)
    ▷ add transition from preceding strategy state to the current strategy state. The guard
    is active if all the rules have been applied in the previous strategy transition.
34:         agentTtransitions.Push(previousState, applyChangesState, IsNotApplyAllRules)
    ▷ add transition from preceding strategy state to state where changes produced by rules
    are applied. The guard is active if not all rules have been applied in the previous
    strategy transition
35:     else
36:         AddTtransition(previousState, strategy.Name, strategy)
    ▷ add transition from preceding strategy state to the current strategy state
37:         agentTtransitions.Push(previousState, applyChangesState, IsApplyStructureRule)
    ▷ add transition from preceding state strategy to state in which changes produced by the
    applied rules are committed. The guard is active when the structural rule has been
    applied on the previous strategy transition
38:     end if
39:     end if
40:     end if
41:     end for
42:     agentStates.Push(applyChangesState)
    ▷ adding state in which changes produced by the applied rules are committed
43:     agentTtransitions.Push(applyChangesState, receiveState)
    ▷ adding transition on which changes produced by the applied rules are committed
44:     agentStates.Push(receiveState)
    ▷ add state that receives objects sent by applying the communication rules in other membranes
45:     agentTtransitions.Push(receiveState, s0State)
    ▷ add transition that receives objects sent by applying the communication rules in other membranes
46:     agentStates.Push(s0State)
    ▷ add an intermediary state
47:     agentTtransitions.Push(s0State, endState, IsNotApplyStructureRule)
    ▷ add transition to the final state in which nothing happens unless a structural rule was applied
48:     agentTtransitions.Push(s0State, endState, IsApplyStructureRule)
    ▷ add the transition to the final state on which structural changes are made if the structure rule has
    been applied
49:     agentStates.Push(endState)
    ▷ add the final state
50:     agentsStates.PushAll(agentStates.Content())
    ▷ add the contents of the stack that holds the current agent states to the stack that holds the states
    of all agents
51:     agentsTtransitions.PushAll(agentStates.Content())
    ▷ add the contents of the stack that holds the current agent transitions to the stack that holds the
    transitions of all agents
52: end for

```

Table 6: Multiset rules ($R_{pulsing}$) of the SP systems model of the pulsing cell.

Rule	Kinetic constant
r_1 : PluxL_geneLuxR $\xrightarrow{k_1}$ PluxL_geneLuxR + rnaLuxR_RNAP	$k_1 = 0.1$
r_2 : rnaLuxR_RNAP $\xrightarrow{k_2}$ rnaLuxR	$k_2 = 3.2$
r_3 : rnaLuxR $\xrightarrow{k_3}$ rnaLuxR + proteinLuxR_Rib	$k_3 = 0.3$
r_4 : rnaLuxR $\xrightarrow{k_4}$	$k_4 = 0.04$
r_5 : proteinLuxR_Rib $\xrightarrow{k_5}$ proteinLuxR	$k_5 = 3.6$
r_6 : proteinLuxR $\xrightarrow{k_6}$	$k_6 = 0.075$
r_7 : proteinLuxR + signal30C6 $\xrightarrow{k_7}$ proteinLuxR_30C6	$k_7 = 1.0$
r_8 : proteinLuxR_30C6 $\xrightarrow{k_8}$	$k_8 = 0.0154$
r_9 : proteinLuxR_30C6 + proteinLuxR_30C6 $\xrightarrow{k_9}$ LuxR2	$k_9 = 1.0$
r_{10} : LuxR2 $\xrightarrow{k_{10}}$	$k_{10} = 0.0154$
r_{11} : LuxR2 + PluxR_geneCI $\xrightarrow{k_{11}}$ PluxR_LuxR2_geneCI	$k_{11} = 1.0$
r_{12} : PluxR_LuxR2_geneCI $\xrightarrow{k_{12}}$ LuxR2 + PluxR_geneCI	$k_{12} = 1.0$
r_{13} : PluxR_LuxR2_geneCI $\xrightarrow{k_{13}}$ PluxR_LuxR2_geneCI + rnaCI_RNAP	$k_{13} = 1.4$
r_{14} : rnaCI_RNAP $\xrightarrow{k_{14}}$ rnaCI	$k_{14} = 3.2$
r_{15} : rnaCI $\xrightarrow{k_{15}}$ rnaCI + proteinCI_Rib	$k_{15} = 0.3$
r_{16} : rnaCI $\xrightarrow{k_{16}}$	$k_{16} = 0.04$
r_{17} : proteinCI_Rib $\xrightarrow{k_{17}}$ proteinCI	$k_{17} = 3.6$
r_{18} : proteinCI $\xrightarrow{k_{18}}$	$k_{18} = 0.075$
r_{19} : proteinCI + proteinCI $\xrightarrow{k_{19}}$ CI2	$k_{19} = 1.0$
r_{20} : CI2 $\xrightarrow{k_{20}}$	$k_{20} = 0.00554$
r_{21} : LuxR2 + PluxPR_geneGFP $\xrightarrow{k_{21}}$ PluxPR_LuxR2_geneGFP	$k_{21} = 1.0$
r_{22} : PluxPR_LuxR2_geneGFP $\xrightarrow{k_{22}}$ LuxR2 + PluxPR_geneGFP	$k_{22} = 1.0$
r_{23} : LuxR2 + PluxPR_CI2_geneGFP $\xrightarrow{k_{23}}$ PluxPR_LuxR2_CI2_geneGFP	$k_{23} = 1.0$
r_{24} : PluxPR_LuxR2_CI2_geneGFP $\xrightarrow{k_{24}}$ LuxR2 + PluxPR_CI2_geneGFP	$k_{24} = 1.0$
r_{25} : CI2 + PluxPR_geneGFP $\xrightarrow{k_{25}}$ PluxPR_CI2_geneGFP	$k_{25} = 5.0$
r_{26} : PluxPR_CI2_geneGFP $\xrightarrow{k_{26}}$ CI2 + PluxPR_geneGFP	$k_{26} = 0.0000001$
r_{27} : CI2 + PluxPR_LuxR2_geneGFP $\xrightarrow{k_{27}}$ PluxPR_LuxR2_CI2_geneGFP	$k_{27} = 5.0$
r_{28} : PluxPR_LuxR2_CI2_geneGFP $\xrightarrow{k_{28}}$ CI2 + PluxPR_LuxR2_geneGFP	$k_{28} = 0.0000001$
r_{29} : PluxPR_LuxR2_geneGFP $\xrightarrow{k_{29}}$ PluxPR_LuxR2_geneGFP + rnaGFP_RNAP	$k_{29} = 4.0$
r_{30} : rnaGFP_RNAP $\xrightarrow{k_{30}}$ rnaGFP	$k_{30} = 3.36$
r_{31} : rnaGFP $\xrightarrow{k_{31}}$ rnaX + proteinGFP_Rib	$k_{31} = 0.667$
r_{32} : rnaGFP $\xrightarrow{k_{32}}$	$k_{32} = 0.04$
r_{33} : proteinGFP_Rib $\xrightarrow{k_{33}}$ proteinGFP	$k_{33} = 3.78$
r_{34} : proteinGFP $\xrightarrow{k_{34}}$	$k_{34} = 0.0667$

Table 7: Multiset rules ($R''_{pulsing}$) of the kP systems model of the pulsing cell.

Rule
r_1 : PluxL_geneLuxR \rightarrow PluxL_geneLuxR + rnaLuxR_RNAP
r_2 : proteinLuxR \rightarrow
r_3 : proteinLuxR + signal30C6 \rightarrow proteinLuxR_30C6
r_4 : proteinLuxR_30C6 \rightarrow
r_5 : proteinLuxR_30C6 + PluxPR_geneGFP \rightarrow PluxPR_LuxR2_geneGFP
r_6 : PluxPR_LuxR2_geneGFP \rightarrow PluxPR_LuxR2_geneGFP + proteinGFP
r_7 : proteinGFP \rightarrow
r_8 : signal30C6 \rightarrow signal30C6 (pulsing)

Notes on Spiking Neural P Systems with Structural Plasticity Computing Morphisms

Francis George C. Cabarle, Henry N. Adorna

Algorithms & Complexity Lab
Department of Computer Science
UP Diliman, 1101, Quezon City, Philippines
email: fccabarle@up.edu.ph, hnadorna@dcs.upd.edu.ph

Abstract. Spiking Neural P Systems with Structural Plasticity (or SPSNP systems) introduced the biological feature of structural plasticity into SNP systems: neurons can create or remove synapses among them. This work is an ongoing effort to investigate SPSNP transducers for computing morphisms.

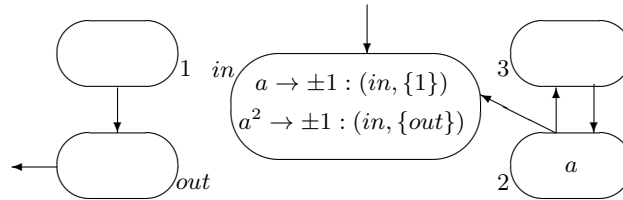
Key words: Spiking Neural P systems, structural plasticity, transducers, morphisms

SPSNP systems were introduced in [1], where plasticity rules are added to the standard spiking rules in SNP systems literature (more information can be found in [2] and [4]). Plasticity rules allow the creation and removal of synapses, therefore changing the synapse graph of the system. The addition of plasticity rules is a partial response to the open problem **D** by Păun in [4] where “dynamism for synapses” is proposed. Spiking and forgetting rule syntax and semantics from SNP systems remain the same for SPSNP systems. The plasticity rules are of the form $E/a^c \rightarrow \alpha k : (i, N_j)$, where: E is a regular expression over $\{a\}$; c spikes are consumed when the rule is applied in neuron σ_i ; $\alpha \in \{+, -, \pm, \mp\}$ indicates whether to create, delete, create-then-delete, or delete-then-create synapses, respectively; N_j is a set of neurons that σ_i can create synapses to or delete synapses from. Every time σ_i creates a synapse to some $\sigma_k, k \in N_j$, one spike is immediately sent from σ_i to σ_k . For example, applying the rule $a \rightarrow +1 : (i, \{j, k\})$ in σ_i means consuming one spike and creating one synapse from σ_i nondeterministically to either σ_j or σ_k .

In [3] SNP transducers were investigated: SNP transducers are simply SNP systems with both input and output neurons. SNP transducers were used to compute morphisms from three classes over a binary alphabet: (non)erasing, (non)length preserving, (not) powers of the same word. Consider the morphism $\mu : \{0, 1\}^* \rightarrow \{0, 1\}^+$ where $\mu(0) = 01$ and $\mu(1) = 10$. From [3] we have that μ is a morphism that is non-erasing, non-length preserving, and not power of the same word. It is known that morphisms such as μ cannot be computed by SNP transducers (see [3] for more information). What follows is an overview of an SPSNP transducer computing μ using an encoding for the input spike train.

Consider the SPSNP transducer Π_μ , graphically shown below. The initial configuration is where only σ_2 contains one spike. In the figure, neurons with no rules shown have only the rule $a \rightarrow a$, and we omit this from writing. We mark the time for the initial configuration as time 0. The infinite spike train that is considered after time 0

will be the input of Π_μ . Recall that no spike corresponds to an input (or output) of 0, and a spike corresponds to an input (or output) of 1. We use an encoding γ on an input $w \in \{0, 1\}^* \cup \{0, 1\}^\omega$ of μ for Π_μ as follows: only the inputs entering Π_μ at odd times are considered as correct or valid inputs. For example, an input 011010 for μ is encoded as $\gamma(011010) = \mathbf{0}^{(1)}\mathbf{0}^{(2)}\mathbf{1}^{(3)}\mathbf{0}^{(4)}\mathbf{1}^{(5)}\mathbf{0}^{(6)}\mathbf{0}^{(7)}\mathbf{0}^{(8)}\mathbf{1}^{(9)}\mathbf{0}^{(10)}\mathbf{0}^{(11)}$, where $(t), t \in \{1, 2, \dots, 11\}$, is the time when no spike or a spike enters Π_μ . Whenever t is even this corresponds to an input we do not consider. In particular, an input of 0 or no spike must be present on even times, so that these inputs do not interfere with the functioning of Π_μ . Only the inputs for odd values of t , shown in bold numbers, are considered.



The correct output of Π_μ is only considered starting at time 3, so that we have a fixed and unambiguous prefix 0^2 in the output, i.e., given an input w , we have $\Pi_\mu(\gamma(w)) = 0^2\mu(w)$. Neuron *in* is the only neuron with plasticity rules, and acts similar to a switch: it activates (by sending one spike) either σ_1 or σ_{out} depending on whether σ_{in} contains one or two spikes, respectively. In the case when no spike enters the system (an input of 0) at some time t then, the plasticity rule with $E = a$ is always applied in σ_{in} . Applying this plasticity rule creates a synapse (which is deleted in the next step) and sends one spike from σ_{in} to σ_1 . Therefore at time $t + 1$ no spike is sent to the environment, and σ_1 sends one spike to σ_{out} . At time $t + 2$ one spike is sent to the environment by σ_{out} . We therefore produced an output produced by $\mu(0)$. If however we have a spike as input at time t (input of 1), then the plasticity rule with $E = a^2$ is always applied. Applying this rule creates a synapse (which is deleted in the next step) and sends one spike from σ_{in} to σ_{out} . At time $t + 1$ we have one spike sent to the environment by σ_{out} , and no spike is sent at time $t + 2$. We therefore produced an output produced by $\mu(1)$.

References

1. Cabarle, F.G.C., Adorna, H.N.: Spiking Neural P Systems with Structural Plasticity. ACMC 2013
2. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae, vol. 71,(2,3) pp. 279-308 (2006)
3. Păun, Gh., Pérez-Jiménez, M.J., Rozenberg, G.: Computing Morphisms by Spiking Neural P Systems. IJFC. vol. 8(6) pp. 1371-1382 (2007)
4. Păun, Gh., Pérez-Jiménez, M.J.: Spiking Neural P Systems. Recent Results, Research Topics. A. Condon et al. (eds.), Algorithmic Bioprocesses, Springer (2009)

The Abilities of P Colony Based Models in Robot Control

Luděk Cienciala, Lucie Ciencialová, Miroslav Langer, and Michal Perdek

Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic
{ludek.cienciala, lucie.ciencialova, miroslav.langer, michal.perdek}@fpf.slu.cz

Abstract. P colonies were introduced in 2004 (see [10]) as an abstract computing device composed of independent single membrane agents, reactively acting and evolving in a shared environment. Each agent is equipped with set of rules which are structured into simple programs. There are some models outgoing from original P colony. In most cases, models are equipped with such components which are associated with the environment. This is the case of Pcol automaton too. The agents work not only with environment but also with an input tape. We use this two theoretical computational devices to build complex robot controllers. In this paper we introduce simple controllers one fulfilling instructions and the other one for passing the maze using right-hand rule. We followed two different approaches and ideas and we present here the obtained results.

1 Introduction

Recently, the robotics has been more and more expanding and intervening in various branches of science like biology, psychology, genetics, engineering, cognitive science, neurology etc. An effort to create robots with an artificial intelligence which are able to cogitate or to solve various types of problems refers to hardware and software limits. Many of these limits are managed to be eliminated by the interdisciplinary approach which allows creating new concepts and technics suitable for the robot control and facture of the new hardware.

The robot control is often realized by the classical procedures known from the control theory (see [17]), concepts inspired by the biology, evolution concepts (see [7]) or with use of the decentralized approaches (see [16]).

Another proposal to use the P systems as an instrument for the realization of the robot control can be found in [1], [2]. The controller based on the numerical P systems allows the parallelization of the computation of the function which the controller has to perform. In [18] the authors showed the use of P systems for modelling mobile robot path planning. The authors proposed a modified membrane-inspired algorithm based on particle swarm optimization, which combines membrane systems with particle swarm optimization.

The robot's control is realized by the control unit. Robots are equipped with the various types of sensors, cameras, gyroscopes and further hardware which all

together represent the robots perception. These hardware components provide to the control unit the information about the actual state of the environment in which the robot is present and also the information about the internal states of the robot. After the transformation of these inputs a new data is generated which is forwarded to the actuators like the wheels, robotic arm etc. Thus the robot can pass the obstacle by using the sensors and adjusting the speed of the particular wheels. So the objective of the control unit is to transform input signals to the output signals which consequently affect the behaviour of the robot. Transformation of these signals can be done computationally in various ways with use of the knowledge or fuzzy knowledge systems, artificial neural networks, or just with use of the membrane systems namely P colonies as it will be shown in this paper.

P colonies were introduced in 2004 as an abstract computing devices composed from independent single membrane agents, reactively acting and evolving in a shared environment ([9]). P colonies reflect motivation from colonies of grammar systems, i.e. the idea of the devices composed from as simple as possible agents placed in a common environment; the system, which produce nontrivial emergent behaviour, using the environment only as the communication medium with no internal rules. P colonies consist of single membrane agents, cells, “floating” in a common environment. Sets of rules of cells are structured to simple programs in P colonies. Rules in a program have to act in parallel in order to change all objects placed into the cell, in one derivation step. Objects are grouped into cells or they can appear in their completely passive environment in which these cells act. In [9] the set of programs was extended by the checking rules. These rules give an opportunity to the agents to opt between two possibilities. They have form r_1/r_2 . If the checking rule is performed, the rule r_1 has higher priority to be executed than the rule r_2 has. It means that the agent checks the possibility to use rule r_1 . If it can be executed, the agent has to use it. If the first rule cannot be applied, the agent uses the second one.

Pcol automaton was introduced in order to describe the situation, when P colonies behave according to the direct signals from the environment (see [3]). This modification of the P colony is constructed in order to recognize input strings. In addition to the writing and communication rules usual for a P colony cells in Pcol automata have also tape rules. Tape rules are used for reading next symbol from the input tape and changing an object in cell(s) to the read symbol. Depending on the way tape rules and other rules can take a part in derivation process several computation modes are treated. After reading the whole input word, computation ends with success if the Pcol automaton reaches one of its accepting configurations. So, in accordance with finite automata, Pcol automata are string accepting devices based on the P colony computing mechanisms. Controller based on P colonies described in this paper is drawn up as a group of cooperating agents who live in shared environment, through which agents can communicate. Such controller can be used for wide range of tasks associated with control issues.

In this paper we follow two ideas of how to control robot with use of P colonies. The first controller model uses Pcol automaton for which we put on the input tape the instruction for robot. The agents have to read the current information from the tape and with the objects in the environment that come from receptors they generate objects - commands for actuators. The second idea is to use original model of P colony and put all information to the environment. We divide the agents into modules that will perform the individual functions in the control of the robot.

The paper is structured as follows: After the brief introduction, we will present formal model of the P colony and of the Pcol automata in section 2.

In section 3 Pcol automaton is used for robot control. Our device consists from 7 agents structured into four natural modules; namely control unit, left actuator controller, right actuator controller and infra-red receptor. The Pcol automaton uses checking rules but it can be redesigned to the Pcol automaton without using checking rules with the greater number of agents then 7. In the section 4 we describe P colony controller which can go through the maze using right-hand rule. The P colony has 5 agents with capacity 3. At the conclusion we compare both models and we outline one of possible directions for further research.

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

2 Preliminaries on the P colonies and Pcol automata

P colonies were introduced in 2004 (see [10]) as an abstract computing device composed of independent single membrane agents, reactively acting and evolving in a shared environment. This model is inspired by structure and function of a community of living organisms in a shared environment.

Each agent is represented by a collection of objects embedded in a membrane. The number of objects inside each agent is constant during the computation. With each agent is associated a set of simple programs. Each program is composed from the rules which can be of two types. The first type of rules, called the evolution rules, are of the form $a \rightarrow b$. It means that the object a inside the agent is rewritten (evolved) to the object b . The second type of rules, called the communication rules, are of the form $c \leftrightarrow d$. If the communication rule is performed, the object c inside the agent and the object d outside the agent swap their places. Thus after executing the rule, the object d appears inside the agent and the object c is placed outside the agent.

If the checking rule r_1/r_2 is performed, then the rule r_1 has higher priority to be executed over the rule r_2 . It means that the agent checks whether the rule r_1 is applicable. If the rule can be executed, then it is compulsory for the agent to use it. If the rule r_1 cannot be applied, then the agent uses the rule r_2 . The program determines the activity of the agent. The agent can change the content of itself or of the environment.

The environment contains several copies of the basic environmental object denoted by e . The environmental object e appears in arbitrary large number of copies in the environment.

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

For more information about P systems see [14] or [15].

Definition 1. *The P colony of the capacity k is a construct*

$$\Pi = (A, e, f, V_E, B_1, \dots, B_n), \text{ where}$$

- A is an alphabet of the colony, its elements are called objects;
- $e \in A$ is the basic object of the colony;
- $f \in A$ is the final object of the colony;
- V_E is a multiset over $A - \{e\}$, it determines the initial state (content) of the environment;
- B_i , $1 \leq i \leq n$, are agents, each agent is a construct $B_i = (O_i, P_i)$, where
 - O_i is a multiset over A , it determines the initial state (content) of the agent, $|O_i| = k$;
 - $P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is a finite multiset of programs, where each program contains exactly k rules, which are in one of the following forms each:
 - * $a \rightarrow b$, called the evolution rule;
 - * $c \leftrightarrow d$, called the communication rule;
 - * r_1/r_2 , called the checking rule; r_1, r_2 are the evolution rules or the communication rules.

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

We will use the parallel model of P colonies for the robot controller. It means that each agent tries to find one usable program in current configuration at each step of the parallel computation. If the number of applicable programs is higher than one, then the agent nondeterministically chooses one of the programs. The maximal possible number of agents is active at each step of the computation. By use of chosen programs the P colony goes from one configuration to another one.

The configuration is called halting if any agent has no applicable program. The result of computation is associated with the halting configuration. It is the number of final objects placed in the environment at the end of computation. Because of nondeterminism in computation we obtain the set of numbers computed by P colony.

By extending the P colony with the input tape we obtain a string accepting/recognizing device; the PCol automaton (see [3]). The input tape contains the input string which can be read by the agents. The input string is the sequence of the symbols. To access the tape the agents use special tape rules (T -rules).

The rules not accessing the tape are called non-tape rules (N -rules). The computation and use of the T -rules is very similar to the use of the rules in the P colonies. Once any of the agents uses its T -rule, the actual symbol on the tape is considered as read.

Definition 2. A PCol automaton of the capacity k and with n agents, $k, n \geq 1$, is a construct

$$\Pi = (A, e, w_E, (O_1, P_1), \dots, (O_n, P_n)), \text{ where}$$

- A is a finite set, an alphabet of the PCol automaton, its elements are called objects;
- e is an environmental object, $e \in A$;
- w_E is a multiset over $A - \{e\}$ defining the initial content of the environment;
- $(O_i, P_i), 1 \leq i \leq n$ is an i -th agent, where
 - O_i is a multiset over A defining the initial content of the agent, $|O_i| = k$;
 - P_i is a finite set of the programs, $P_i = T_i \cup N_i, T_i \cap N_i = \emptyset$; where every program is formed from k rules of the following types:
 - * the tape rules (T -rules for short)
 - $a \xrightarrow{T} b$ are called the rewriting T -rules, $a, b \in A$;
 - $a \xleftrightarrow{T} b$ are called the communication T -rules, $a, b \in A$;
 - * the non-tape rules (N -rules for short)
 - $a \rightarrow b$ are called the rewriting N -rules, $a, b \in A$;
 - $a \leftrightarrow b$ are called the communication N -rules, $a, b \in A$;
 - r_1/r_2 are called the checking N -rules, r_1, r_2 are rewriting or communication rules;
 - * T_i is a finite set of tape programs (T -programs for short) consisting from one T -rule and $k - 1$ N -rules;
 - * N_i is a finite set of non-tape programs (N -programs for short) consisting from k N -rules.

The configuration of the PCol automaton is the $(n + 2)$ -tuple $(v_T; v_E; v_1, \dots, v_n)$, where $v_T \in A^*$ the unprocessed (unread) part of the input string, $v_E \in (A - \{e\})^*$ is a multiset of the objects different from the object e placed in the environment of the PCol automaton and $v_i, 1 \leq i \leq n$ is a content of the i -th agent.

The computation starts in the initial configuration defined by the input string ω on the tape, the initial content of the environment w_E and the initial content of the agents $O_i, 1 \leq i \leq n$. The computation is performed similarly as in the case of P colonies. When an agent looks for applicable program, it scans the set of T -programs at first. If there is at least one applicable program, agent use one of applicable programs. If there is no suitable T -program, agent proceed with the set of N -programs. The current symbol on the input tape we consider as read iff at least one agent uses its T -program in the particular derivation step.

The computation halts if no agent has any applicable program. The halting configuration is accepting iff all symbols on the input tape are already read. In such a case we can say that Pcol automaton accepts input word ω if there exists at least one accepting halting computation starting in the initial configuration with ω on the input tape.

3 Pcol automaton controller with commands on the input tape

The main advantage of using PCol automaton in the controlling robot behavior is the parallel proceeding of the data done by very primitive computational units using very simple rules.

Combining the modularity and the PCol automaton we obtain a powerful tool to control robot behaviour. PCol automaton is a parallel computation device. Collaterally working autonomous units sharing common environment provide a fast computation device. Dividing agents into the modules allows us to compound agents controlling single robot sensors and actuators. All the modules are controlled by the main controlling unit. Input tape gives us an opportunity to plan robot actions. Each input symbol represents a single instruction which has to be done by the robot, so the input string is the sequence of the actions which guides the robot to reach its goal; performing all the actions. In this meaning the computation ends by halting, and it is successful if whole input tape is read. For this purpose we use special symbol which marks the end of the actions. When the last tape symbol is read, the controlling unit sends halting symbol to the receptor modules, which stop their action. All the other modules will stop their actions independently on the halting symbols, because they will not obtain any other instruction from the controlling unit.

We construct a PCol automaton with capacity 2 and 7 agents. The agents are divided into the four modules: *Control unit* contains two agents, *Left actuator controller* and *Right actuator controller*, each contains one agent and *Infra-red receptor* contains three agents. Entire automaton is amended by the *input* and *output filter*. The input filter codes signals from the robots receptors and spread the coded signal into the environment. In the environment there is the coded signal used by the agents. The output filter decodes the signal from the environment which the actuator controllers sent into it. Decoded signal is forwarded to the robots actuators.

If the program is formed from rules of the same kind we can write $\langle ab \rightarrow cd \rangle$ or $\langle ab \leftrightarrow cd \rangle$ instead of $\langle a \rightarrow c; b \rightarrow d \rangle$ or $\langle a \leftrightarrow c; b \leftrightarrow d \rangle$. Let us introduce the formal specification of the mentioned PCol automaton. We will not present all the programs entirely, but we focus on the individual sets of programs and their functions.

We define Pcol automaton as follows:

$$\begin{aligned} \Pi &= (A, e, w_E, (O_1, P_1), \dots, (O_7, P_7)), \text{ where} \\ A &= \{0_L, 0_R, 1_L, 1_R, e, F_F, \overline{F_F}, F_L, \overline{F_L}, F_R, \overline{F_R}, G_F, G_L, G_R, I_F, \\ &\quad I_L, I_R, M_F, M_L, M_R, N_F, \overline{N_F}, N_L, \overline{N_L}, N_R, \overline{N_R}, R_T, W_F, \\ &\quad W_L, W_R, W_T, H\}, \\ w_E &= \{\varepsilon\}. \end{aligned}$$

Let us describe the meaning of the particular objects:

$0_L, 0_R$	Signal for the output filter - don't move the left/right wheel.
$1_L, 1_R$	Signal for the output filter - move the left/right wheel.
$\overline{F_F}, \overline{F_L}, \overline{F_R}$	Signal from the input filter - no obstacle in front/on the left/right.
$\overline{N_F}, \overline{N_L}, \overline{N_R}$	Signal from the input filter - obstacle in front/on the left/right.
F_F, F_L, F_R	Signal from the IR module to the control unit- no obstacle in front/on the left/right.
N_F, N_L, N_R	Signal from the IR module to the control unit- obstacle in front/on the left/right.
I_F, I_L, I_R	Signal from the control unit to the IR module - is there an obstacle in front/on the left/right?
M_F, M_L, M_R	Signal from the control unit to the actuator controllers - move front, turn left/right.
R_T	Signal from the actuator controllers to the control unit - read next tape symbol.
H	Halting symbol - the last symbol on the tape.

Remaining objects (W_F, W_L, W_R, G) are used for inner representation of the actions and as the complementary objects.

Particular modules and agents which they contain are defined as follows:

Control unit: The control unit has two agents B_1 and B_2 with the initial contents $O_1 = eR_T$ and $O_2 = eR_T$. They process the input string in parallel. The first programs are designed for reading instruction from the input tape.

$$\begin{aligned}
 P_1 : & \langle R_T \xrightarrow{T} M_F; e \rightarrow e \rangle - \text{instruction "move front"}; \\
 & \langle R_T \xrightarrow{T} M_L; e \rightarrow e \rangle - \text{instruction "move left"}; \\
 & \langle R_T \xrightarrow{T} M_R; e \rightarrow e \rangle - \text{instruction "move right"}; \\
 & \langle R_T \xrightarrow{T} H; e \rightarrow H \rangle - \text{instruction "halt"}; \\
 P_2 : & \langle R_T \xrightarrow{T} M_F; e \rightarrow I_F \rangle; \\
 & \langle R_T \xrightarrow{T} M_L; e \rightarrow I_L \rangle; \\
 & \langle R_T \xrightarrow{T} M_R; e \rightarrow I_R \rangle; \\
 & \langle R_T \xrightarrow{T} H; e \rightarrow H \rangle;
 \end{aligned}$$

The second agent generates the object to be send to infra-red modules. This object has information about the next move the robot has to do. The following programs are to send this object of type I_x to the environment; $x \in \{F, L, R\}$.

$$P_2 : \langle I_x \leftrightarrow e; M_x \rightarrow W_x \rangle$$

There are two different types of symbols which can appear in the environment after some steps - F_x the space in requested direction is free or N_x there is obstacle in requested direction. The agent B_2 consumes the object and if the object is of the type F_x it generates the instruction for agent B_1 to generate command for actuators.

$$\begin{aligned}
 P_2 : & \langle W_x \rightarrow e; e \leftrightarrow F_x/e \leftrightarrow N_x \rangle; \\
 & \langle F_x \rightarrow G_x; e \rightarrow e \rangle; \\
 & \langle G_x \leftrightarrow e; e \rightarrow W_T \rangle
 \end{aligned}$$

The object of type G_x is processed by the agent B_1 . After consuming G_x it generates two objects M_x , the information for actuator module to generate commands for wheels.

$$\begin{aligned} P_1 : & \langle e \leftrightarrow G_x; M_x \rightarrow M_x \rangle; \\ & \langle G_x M_x \rightarrow M_x M_x \rangle; \\ & \langle M_x M_x \leftrightarrow ee \rangle; \\ & \langle ee \rightarrow eW_T \rangle \end{aligned}$$

When inside both agents there are objects W_T it means that agents have to wait for execution of their command. It is performed before object R_T appears in the environment. They are generated by actuator module after sending the command for the wheels to the environment.

$$\begin{aligned} P_1 : & \langle W_T \rightarrow e; e \leftrightarrow R_T \rangle \\ P_2 : & \langle W_T \rightarrow e; e \leftrightarrow R_T \rangle \end{aligned}$$

Now the control unit is prepared to read new symbol from the input tape. If the symbol read from the input tape is H , both agents generate a pair of H s. These objects are needed for halting three agents in infra-red module.

$$\begin{aligned} P_1 : & \langle H \rightarrow H; H \leftrightarrow e \rangle \\ P_2 : & \langle HH \leftrightarrow ee \rangle \end{aligned}$$

Infra-red unit: The infra-red module is composed from three agents with similar programs. Each agent is to utilize object coming from the infra-red sensors $\overline{F}_x; x \in \{F, L, R\}$. Each one for one direction (F - front, L - left, R - right). The initial content of all agents in infra-red module is $ee - O_i = ee; i = \{3, 4, 5\}$. If there is object \overline{F}_x or \overline{N}_x in the environment - comes from infra-red sensors, the corresponding agent consumes it. It remakes the object F_x or N_x . If the control unit needs information about situation in the direction x - object I_x is present in the environment. The infra-red unit exchange object F_x or N_x for I_x . If there is no request from control unit, the agents shred the information (they rewrite object F_x or N_x to e). If the object H appears in the environment each agent consumes it preferably and the agent halts.

$$\begin{aligned} P_i = \{ & \langle e \leftrightarrow H/e \leftrightarrow \overline{F}_x; e \rightarrow F_x \rangle; \\ & \langle F_x \leftrightarrow I_x/F_x \rightarrow e; \overline{F}_x \rightarrow e \rangle; \\ & \langle e \leftrightarrow H/e \leftrightarrow \overline{N}_x; e \rightarrow N_x \rangle; \\ & \langle N_x \leftrightarrow I_x/N_x \rightarrow e; \overline{N}_x \rightarrow e \rangle; \\ & \langle I_x e \rightarrow ee; \rangle; \quad i = \{3, 4, 5\} \end{aligned}$$

The left and right actuator controllers wait for the activating signal (M_x) from the control unit. After obtaining the activating signal the controllers try to provide demanded action by sending special objects - coded signal for the output filter into the environment. When the action is performed successfully the actuators send the announcement of the successful end of the action to the control unit.

The left actuator controller:

$$\begin{aligned} O_6 = \{ & e, e \}, \\ P_6 = \{ & \langle e \leftrightarrow M_F; e \rightarrow 1_L \rangle; \langle M_F \rightarrow R_T; 1_L \leftrightarrow e \rangle; \\ & \langle e \leftrightarrow M_R; e \rightarrow 1_L \rangle; \langle M_R \rightarrow R_T; 1_L \leftrightarrow e \rangle; \\ & \langle e \leftrightarrow M_L; e \rightarrow 0_L \rangle; \langle M_L \rightarrow R_T; 0_L \leftrightarrow e \rangle; \\ & \langle R_T \leftrightarrow e; e \rightarrow e \rangle \}. \end{aligned}$$

The right actuator controller:

$$\begin{aligned}
O_7 &= \{ e, e \}, \\
P_7 &= \{ \langle e \leftrightarrow M_F; e \rightarrow 1_R \rangle; \langle M_F \rightarrow R_T; 1_R \leftrightarrow e \rangle; \\
&\quad \langle e \leftrightarrow M_R; e \rightarrow 0_R \rangle; \langle M_R \rightarrow R_T; 0_R \leftrightarrow e \rangle; \\
&\quad \langle e \leftrightarrow M_L; e \rightarrow 1_R \rangle; \langle M_L \rightarrow R_T; 1_R \leftrightarrow e \rangle; \\
&\quad \langle R_T \leftrightarrow e; e \rightarrow e \rangle \}.
\end{aligned}$$

The robot driven by this very simple PCol automaton is able to follow the instruction on the tape safely without crashing into any obstacle. If the instruction cannot be proceeded, the robot stops. This solution is suitable for known robots environment. If the environment is changed before or during the journey and the robot cannot reach the final place it will not crash, either. We consider the computation as successful if the input string is processed and the robot fulfills the last action. The computation is unsuccessful otherwise.

4 P colony robot controller

We construct a P colony with four modules: *Controlling unit*, *Left actuator controller*, *Right actuator controller* and *Infra-red receptor*. Entire colony is amended by the *input* and *output filter*. The input filter codes signals from the robots receptors and spread the coded signal into the environment. In the environment there is the coded signal used by the agents. The output filter decodes the signal from the environment which the actuator controllers sent into it. Decoded signal is forwarded to the robots actuators.

Let us introduce the formal definition of the mentioned P colony:

$$\begin{aligned}
\Pi &= (A, e, V_E, (O_1, P_1), \dots, (O_5, P_5), \emptyset), \text{ where} \\
A &= \{1_L, 1_R, -1_L, -1_R, A_L, A_R, F, F_F, F_O, F_F^F, F_O^F, G, I_F, I_R, H, H_1, L, M_F, \\
&\quad R, R_F, R_O, R_F^F, R_O^F, W_i\} \\
V_E &= \{e\},
\end{aligned}$$

Let us describe the meaning of the particular objects:

- $1_L, -1_L$ Signal for the output filter - move the left wheel forward/backward.
- $1_R, -1_R$ Signal for the output filter - move the right wheel forward/backward.
- F, L, R Signal from the control unit to the actuator controllers - move forward, turn left/right.
- F_F^F, R_F^F Signal from the input filter - no obstacle in front/on the right.
- F_O^F, R_O^F Signal from the input filter - obstacle in front/on the right.
- F_F, R_F Signal from the IR module to the control unit - no obstacle in front/on the right.
- F_O, R_O Signal from the IR module to the control unit - obstacle in front/on the right.
- I_F, I_R Signal from the control unit to the IR module - is there an obstacle in front/on the right?
- G Maze exit.
- H Halting symbol.

Remaining objects (A_L, A_R, H_1, \dots) are used for inner representation of the actions and as complementary objects.

Particular modules and agents which they contain are defined as follows:

Control unit: The control unit ensures the computation. It controls the behaviour of the robot, it asks the data from the sensors and it sends instructions to the actuators by sending particular symbols to the environment. The controlling unit contains set of programs which provides fulfilling set goal, in this case pass through the maze using the right-hand rule. If the exit from the maze is found; symbol G appears in the environment, the control unit releases into the environment special symbol H , which stops the infra-red receptors and the P colony stops and so the robot.

The control unit is realized by one agent B_1 . Initial content of the agent is $O_1 = I_F I_R W_i$. At first the agent sends the request for information about obstacles to the environment.

$$P_1 : \langle I_F I_R \leftrightarrow ee; W_i \rightarrow W_i \rangle$$

Then the agent waits until it receive the answer form infra-red modules.

$$P_1 : \langle W_i \rightarrow W_i; ee \leftrightarrow R_F F_F \rangle; \langle W_i \rightarrow W_i; ee \leftrightarrow R_O F_F \rangle; \\ \langle W_i \rightarrow W_i; ee \leftrightarrow R_O F_O \rangle; \langle W_i \rightarrow W_i; ee \leftrightarrow R_F F_O \rangle;$$

Based on the information about obstacles agent generates commands to actuator units and sends objects to the environment.

$$P_1 : \langle R_O F_O e \rightarrow L L e \rangle; \langle R_F F_F e \rightarrow R R M_F \rangle; \\ \langle R_O F_F e \rightarrow F F e \rangle; \langle R_F F_O e \rightarrow R R M_F \rangle; \\ \langle L L \leftrightarrow ee; e \rightarrow e \rangle; \langle R R \leftrightarrow ee; M_F \rightarrow M_F \rangle; \\ \langle F F \leftrightarrow ee; e \rightarrow e \rangle; \langle M_F e e \rightarrow F F e \rangle$$

After emitting object F, L or R to the environment the content of the agent is eee and agent B_1 has to prepare for the next step. It consumes objects A_L and A_R - the information about successfully performed motion and consumes G or e from the environment. If there is G on the environment, it means that the robot finds the exit from the maze and computation can halt. If there is no G in the environment, the control unit must generate new request.

$$P_1 : \langle ee \leftrightarrow A_L A_R; e \leftrightarrow G/e \rightarrow e \rangle; \\ \langle A_L A_R e \rightarrow I_F I_R W_i \rangle; \\ \langle A_L A_R G \rightarrow H H H \rangle; \\ \langle H H \leftrightarrow ee; H \rightarrow H_1 \rangle$$

Control unit contains program which controls robots behaviour. According to the data obtained from the IR module it sends instructions to the Actuator controllers. It passes the maze using the right-hand rule until it finds symbol G which represents exit from the maze. While it finds the exit the Control unit releases symbol H into the environment which stops the computation.

The actuator controllers wait for the activating signal from the control unit. After obtaining the activating signal the controllers try to provide demanded

action by sending special objects - coded signal for the output filter into the environment. When the action is performed successfully the actuators send the announcement of the successful end of the action to the control unit, the announcement of the unsuccessful end of the action otherwise.

Left Actuator controller:

$$O_2 = \{ e, e, e \},$$

$$P_2 = \{ \langle e \leftrightarrow F; e \rightarrow 1_L; e \rightarrow A_L \rangle; \\ \langle F \rightarrow e; 1_L \leftrightarrow e; A_L \leftrightarrow e \rangle; \\ \langle e \leftrightarrow R; e \rightarrow 1_L; e \rightarrow A_L \rangle; \\ \langle R \rightarrow e; 1_L \leftrightarrow e; A_L \leftrightarrow e \rangle; \\ \langle e \leftrightarrow L; e \rightarrow -1_L; e \rightarrow A_L \rangle; \\ \langle L \rightarrow e; -1_L \leftrightarrow e; A_L \leftrightarrow e \rangle \}.$$

Right Actuator controller:

$$O_3 = \{ e, e, e \},$$

$$P_3 = \{ \langle e \leftrightarrow F; e \rightarrow 1_R; e \rightarrow A_R \rangle; \\ \langle F \rightarrow e; 1_R \leftrightarrow e; A_R \leftrightarrow e \rangle; \\ \langle e \leftrightarrow R; e \rightarrow -1_R; e \rightarrow A_R \rangle; \\ \langle R \rightarrow e; -1_R \leftrightarrow e; A_R \leftrightarrow e \rangle; \\ \langle e \leftrightarrow L; e \rightarrow 1_R; e \rightarrow A_R \rangle; \\ \langle L \rightarrow e; 1_R \leftrightarrow e; A_R \leftrightarrow e \rangle \}.$$

Right and left Actuator controllers wait for the activating signal from the Control unit. According to signal they move the robot in required direction by sending appropriate signal to the output filter.

The infra-red receptors consume all the symbols released into the environment by the input filter. It releases actual information from the sensors on demand of the control unit. The infra-red receptors remove unused data from the environment.

Front Infra-red module:

$$O_4 = \{ e, e, e \},$$

$$P_4 = \{ \langle e \leftrightarrow H/e \leftrightarrow F_F^F; ee \rightarrow eF_F \rangle; \\ \langle e \leftrightarrow H/e \leftrightarrow F_O^F; ee \rightarrow eF_O \rangle; \\ \langle F_F \leftrightarrow I_F/F_F \leftrightarrow e; F_F^F e \rightarrow ee \rangle; \\ \langle F_O \leftrightarrow I_F/F_O \leftrightarrow e; F_O^F e \rightarrow ee \rangle; \\ \langle I_F ee \rightarrow eee; \rangle \}.$$

Right Infra-red module:

$$O_5 = \{ e, e, e \},$$

$$P_5 = \{ \langle e \leftrightarrow H/e \leftrightarrow R_F^F; ee \rightarrow eR_F \rangle; \\ \langle e \leftrightarrow H/e \leftrightarrow R_O^F; ee \rightarrow eR_O \rangle; \\ \langle R_F \leftrightarrow I_R/R_F \leftrightarrow e; R_F^F e \rightarrow ee \rangle; \\ \langle R_O \leftrightarrow I_R/R_O \leftrightarrow e; R_O^F e \rightarrow ee \rangle; \\ \langle I_R ee \rightarrow eee; \rangle \}.$$

Infra-red modules consume all the symbols send by the input filter into the environment. They send actual data to the Control unit on demand.

Robot driven by this P colony is able to pass through simple mazes that are possible to pass using the right-hand rule.

5 Experimental results

We have used the mobile robotics simulation software Webots for the implementation of our controller. Webots is a visualizing tool which allows us to work with the model of robot and verify the functionality of our controller. The simulators of the P colony and PCol automaton are written in the Java and they processes text files with the specification of the P colony systems. The file contains initial

content of the environment, the definition of the agents, including their rules and initial content, and initial content of the input tape, eventually. Both of these simulating tools are interconnected. That allows us to provide our experiments.

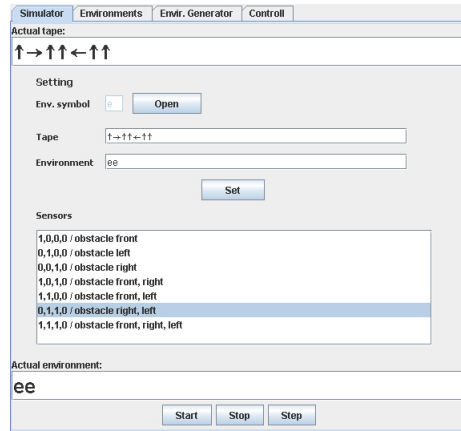


Fig. 1. Simulator environment

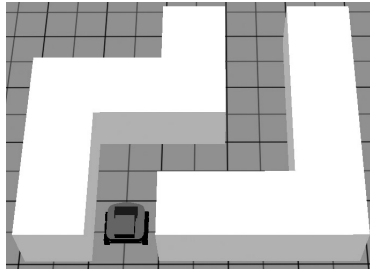


Fig. 2. Starting position

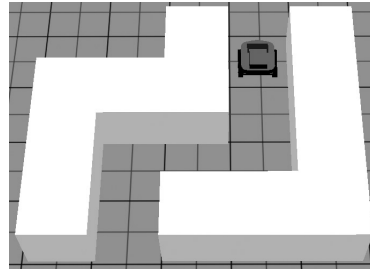


Fig. 3. Ending position

To verify our concept we have provided following simulation. The robot, namely the koala, was placed into the simulating environment in the grid shape with the obstacles. According to the instructions on the input tape (see figure 1) the robot went through prepared maze (see figures 2, 3). All the instructions on the tape were interpreted as performable so the robot went through the maze with use of our controller.

In the second experiment, it was used the P colony controller. A robot passed the maze from the starting position to the final position using the right-hand rule.

Both of the experiments suggest that using these types of controllers on real robots is feasible. Parallel processing of the instructions and the cooperation between the agents appear to be convenient with respect to the possible extension of the controllers by new functions, i.e. by adding new modules.

6 Conclusion and future research

In this paper we have introduced two variants of the robot controller based on the P systems, P colony and Pcol automaton.

We constructed Pcol automaton with seven agents with capacity two that can follow the instructions written on the input tape. The Pcol automaton controller is suitable for use in well-known environment, where the actions can be planned accurately. Unexpected obstacle causes halt of the computation, the goal will not be fulfilled, but the robot will not crash.

We also constructed P colony with five agents with capacity three for passing the maze using right-hand rule. The P colony controller is suitable for the autonomous systems. The behaviour of the robot is given by the rules of the control unit.

The further research can be in following such an idea to combine advantages of both constructed models. Our idea is to construct Pcol automaton with a dynamically changing input tape because the receptors will write the information here. One input symbol can be vector of inputs or we can precompute or filter and scale the values. The active agents will be formed to layers, the first layer is formed from agents that can read the current input symbol, the agents “controlling” behaviour of robot belong to the second layer. The last layer is composed of agents generating output for actuators. The layers are formed from active agents only. So the controlling layer can activate different agents in different situations (guiding closer to the obstacle, avoiding the obstacle or touching and pushing obstacle).

Remark 1. Article has been made in connection with project IT4Innovations Centre of Excellence, reg. no. CZ.1.05/1.1.00/02.0070.

Research was also supported by project OPVK no. CZ.1.07/2.2.00/28.0014 and by projects SGS/24/2013 and SGS/6/2014 .

References

1. Arsene, O., Buiu, C., Popescu, N.: *SNUPS - A simulator for numerical membrane computing*. In: International Journal of Innovative Computing, Information and Control, 7(6), 2011, pp. 3509–3522.
2. Buiu, C., Vasile, C., Arsene, O.: *Development of membrane controllers for mobile robots*, In: Information Sciences, 187, Elsevier Science Inc. New York, NY, USA, 2012, pp. 33–51.

3. Cienciala, L., Ciencialová, L., Csuhaaj-Varjú, E., Vaszil, G.: *PCol Automata: Recognizing Strings with P colonies*. Eight Brainstorming Week on Membrane Computing (V Martínez del Amor, M. A., Păun, G., Hurtado de Mendoza, I. P., Riscon-Núñez, A. (eds.)), Sevilla, 2010, pp. 65–76.
4. Cienciala, L., Ciencialová, L., Langer, M.: *Modularity in P colonies with Checking Rules*. In: Revised Selected Papers 12 th International Conference CMC 2011 (George, M., Păun, Gh., Rozenber, G., Salomaa, A., Verlan, S. eds.), Springer, LNCS 7184, 2012, pp. 104–120.
5. Csuhaaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, G.: *Cells in environment: P colonies*, Multiple-valued Logic and Soft Computing, 12, 3-4, 2006, pp. 201–215.
6. Csuhaaj-Varjú, E., Margenstern, M., Vaszil, G.: *P colonies with a bounded number of cells and programs*. Pre-Proceedings of the 7th Workshop on Membrane Computing (H. J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), Leiden, The Netherlands, 2006, pp. 311–322.
7. Floreano, D. and Mattiussi, C. (2008). *Bio-inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT Press.
8. Freund, R., Oswald, M.: *P colonies working in the maximally parallel and in the sequential mode*. Pre-Proceedings of the 1st International Workshop on Theory and Application of P Systems (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, pp. 49–56.
9. Kelemen, J., Kelemenová, A.: *On P colonies, a biochemically inspired model of computation*. Proc. of the 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, pp. 40–56.
10. Kelemen, J., Kelemenová, A., Păun, Gh.: *Preview of P colonies: A biochemically inspired computing model*. Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE IX (M. Bedau et al., eds.) Boston, Mass., 2004, pp. 82–86.
11. Minsky, M. L.: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
12. Păun, Gh.: *Computing with membranes*. Journal of Computer and System Sciences 61, 2000, pp. 108–143.
13. Păun, Gh.: *Membrane computing: An introduction*. Springer-Verlag, Berlin, 2002.
14. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.
15. P systems web page: <http://psystems.disco.unimib.it>
16. Weiss, G.: *Multiagent systems. A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, Massachusetts, 1999.
17. Wit, C. C., Bastin, G., Siciliano, B.: *Theory of Robot Control*, Springer-Verlag New York, 1996.
18. Wang, X. Y., Zhang, G.X., Zhao, J.B., Rong, H.N., Ipate, F., Lefticaru, R.: *A Modified Membrane-Inspired Algorithm Based on Particle Swarm Optimization for Mobile Robot Path Planning*, International Journal of Computers, Communications & Control, accepted, 2014.

Probabilistic Guarded P systems, a new formal modelling framework

Manuel García-Quismondo, Miguel A. Martínez-del-Amor,
Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mgarciaquismondo@us.es, mdelamor@us.es, marper@us.es

Abstract. Multienvironment P systems are a general, formal framework for modelling the dynamics of population biology, which consists of two main approaches: stochastic and probabilistic. The framework has been successfully used to model biologic systems at both micro (e.g. bacteria colony) and macro (e.g. ecosystems) levels, respectively.

In this paper, we extend the general framework. The extension is made by a new variant within the probabilistic approach, called Probabilistic Guarded P systems (in short, PGP systems). We provide a formal definition, a simulation algorithm to capture the dynamics, and a survey of the associated software.

Keywords: Modelling Framework, Multienvironment P systems, Probabilistic Guarded P systems

1 Introduction

Since P systems were introduced in 1998 [19], they have been utilised as a high level computational modelling framework [10,20]. Their main advantage is the integration of the structural and dynamical aspects of complex systems in a comprehensive and relevant way, while providing the required formalisation to perform mathematical and computational analysis [2].

In this respect, multienvironment P systems are a general formal framework for population dynamics modelling in biology [7]. This framework has two approaches: stochastic and probabilistic. Stochastic approach is usually applied to model *micro*-level systems (such as bacteria colonies), whereas the probabilistic approach is normally used for *macro*-level modelling (real ecosystems, for example). Population Dynamics P systems [2,16,17,3] (PDP systems, in short) are a variant of multienvironment P systems, in the probabilistic approach. PDP systems have been successfully applied to ecological modelling, specially with real ecosystems of some endanger [6,3] and exotic species [3]. PDP systems have shown to comply with four desirable properties of a computational model [2]:

relevance (capture the essential features of the modelled system), *computability* (inherent by P systems), *understandability* (objects and rules capture the dynamics in a simple way), and *extensibility* (rule design is module-oriented).

In this paper, we introduce a brand new variant inside the probabilistic approach of multienvironment P systems: Probabilistic Guarded P systems (*PGP systems*, for short). They are specifically oriented for ecological processes. PGP systems are a computational probabilistic framework which takes inspiration from different Membrane Computing paradigms, mainly from Tissue-like P systems [23], PDP systems [2] and Kernel P systems [12]. This framework aims for simplicity, considering these aspects:

Model designers: In PGP systems, model designers do not need to worry about context consistency. That is to say, they do not need to take into account that all rules simultaneously applied in a cell must define the same polarization in the right-hand side [16]. This is because the framework centralizes all context changes in a single rule per cycle, rather than distributing them across all rules. Therefore, there exist two types of rules: *context-changing* rules and *non context-changing* rules. Due to the nature of the model, only one of such rules can be applied at the same time on each cell, so context inconsistency is not possible. Moreover, the fact that the context is explicitly expressed in each cell and that cells do not contain internal cell structures simplifies transitions between contexts without loss of computational or modelling power.

Simulator developers: The fact that the framework implicitly takes care of context consistency simplifies the development of simulators for these models, as it is a non-functional requirement which does not need to be supported by simulators. In addition, the lack of internal structure in cells simplifies the simulation of object transmission; the model can be regarded as a set of memory regions with no hierarchical arrangement, thus enabling direct region fetching.

Probabilistic Guarded P Systems can be regarded as an evolution of Population Dynamic P systems. In this context, PGP systems propose a modelling framework for ecology in which inconsistency (that is to say, undefined context of membranes) is handled by the framework itself, rather than delegating to simulation algorithms. In addition, by replacing alien concepts to biology (such as electrical polarizations and internal compartment hierarchies) by state variables known as *flags* and defined by designers models are more natural to experts, thus simplifying communication between expert and designer.

This paper is structured as follows. Section 2 introduces some preliminaries. Section 3 shows the formal framework of multienvironment P systems, and the two main approaches. Section 4 describes the framework of PGP systems, providing a formal definition, some remarks about the semantics of the model, and a comparison with other similar frameworks of Membrane Computing. Section 5 provides a simulation algorithm, and a software environment based on P-Lingua and a C++ simulator. Section 6 summarizes an ecosystem under

study with PGP systems. Finally, Section 7 ends the paper with conclusions and future work.

2 Preliminaries

An *alphabet* Γ is a non-empty set whose elements are called *symbols*. An ordered finite sequence of symbols of Γ is a *string* or *word* over Γ . As usual, the empty string (with length 0) will be denoted by λ . The set of all strings over an alphabet Γ is denoted by Γ^* . A *language* over Γ is a subset of Γ^* .

A *multiset* m over an alphabet Γ is a pair $m = (\Gamma, f)$ where $f : \Gamma \rightarrow \mathbb{N}$ is a mapping. For each $x \in \Gamma$ we say that $f(x)$ is the *multiplicity* of the symbol x in m . If $m = (\Gamma, f)$ is a multiset, then its *support* is defined as $\text{supp}(m) = \{x \in \Gamma \mid f(x) > 0\}$. A multiset is finite if its support is a finite set. A *set* is a multiset such that the multiplicity of each element of its support, is equal to 1.

If $m = (\Gamma, f)$ is a finite multiset over Γ , and $\text{supp}(m) = \{a_1, \dots, a_k\}$ then it will be denoted as $m = a_1^{f(a_1)} \dots a_k^{f(a_k)}$ (here the order is irrelevant), and we say that $f(a_1) + \dots + f(a_k)$ is the cardinal of m , denoted by $|m|$. The empty multiset is denoted by \emptyset . We also denote by $M(\Gamma)$ the set of all finite multisets over Γ .

Let $m_1 = (\Gamma, f_1)$ and $m_2 = (\Gamma, f_2)$ multisets over Γ . We define the following concepts:

- The union of m_1 and m_2 , denoted by $m_1 + m_2$ is the multiset (Γ, g) , where $g = f_1 + f_2$, that is, $g(x) = f_1(x) + f_2(x)$ for each $x \in \Gamma$.
- The relative complement of m_2 in m_1 , denoted by $m_1 \setminus m_2$ is the multiset (Γ, g) , where $g = f_1(x) - f_2(x)$ if $f_1(x) \geq f_2(x)$ and $g(x) = 0$ otherwise.

We also say that m_1 is a submultiset of m_2 , denoted by $m_1 \subseteq m_2$, if $f_1(x) \leq f_2(x)$ for each $x \in \Gamma$.

Let $m = (\Gamma, f)$ a multiset over Γ and A a set. We define the intersection $m \cap A$ as the multiset (Γ, g) , where $g(x) = f(x)$ for each $x \in \Gamma \cap A$, and $g(x) = 0$ otherwise.

3 Multienvironment P systems

Definition 1. A *multienvironment P system of degree* (q, m, n) with $q \geq 1$, $m \geq 1$, $n \geq 0$, taking T time units, $T \geq 1$, is a tuple

$$\Pi = (G, \Gamma, \Sigma, \Phi, \mu, T, n = \sum_{j=1}^m n_j, \{\Pi_{k,j} \mid 1 \leq k \leq n_j, 1 \leq j \leq m\}, \{A_j \mid 1 \leq j \leq m\}, \mathcal{R}_E)$$

where:

- $G = (V, S)$ is a directed graph. Let $V = \{e_1, \dots, e_m\}$ whose elements are called *environments*;

- Γ, Σ and Φ are finite alphabets such that $\Sigma \subsetneq \Gamma$ and $\Gamma \cap \Phi = \emptyset$.
- μ is a rooted tree with $q \geq 1$ nodes labelled by elements from $\{1, \dots, q\} \times \{0, +, -\}$.
- $n = \sum_{j=1}^m n_j$, with $n_j \geq 0$
- For each k ($1 \leq k \leq n_j$, $1 \leq j \leq m$), $\Pi_{k,j}$ is a tuple $(\Gamma, \mu, \mathcal{M}_{1,j}^k, \dots, \mathcal{M}_{q,j}^k, \mathcal{R}_j, i_{in})$, where:
 - For each i , $1 \leq i \leq q$, $\mathcal{M}_{i,j}^k \in M(\Gamma)$.
 - \mathcal{R}_j is a finite set of rules of the type: $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$, being $u, v, u', v' \in M(\Gamma)$, $1 \leq i \leq q$, $\alpha, \alpha' \in \{0, +, -\}$ and p is a real computable function whose domain is $\{0, \dots, T\}$.
 - i_{in} is a node from μ .
- Let us note that n_j can be eventually 0, so that there would not exist any $\Pi_{k,j}$ for such an environment j .
- For each j , $1 \leq j \leq m$, $f_j \in \Phi$ and $E_j \in M(\Sigma)$.
- \mathcal{R}_E is a finite set of rules among environments of the types:

$$\begin{array}{ccc} (x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}} & & (\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j_1}} \\ \{f\}(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}} & & \{f\}(u, f)_{e_j} \xrightarrow{p_4} (v, g)_{e_j} \end{array}$$

being $x, y_1, \dots, y_h \in \Sigma$, $(e_j, e_{j_i}) \in S$, $1 \leq j \leq m$, $1 \leq i \leq h$, $1 \leq k \leq n$, $f, g \in \Phi$, $u, v \in M(\Gamma)$ and p_1, p_2, p_3, p_4 are computable functions whose domain is $\{0, \dots, T\}$.

- For each j , $1 \leq j \leq m$, $A_j \in M(\Sigma \cup \Phi)$.

A multienvironment P system of grade (q, m, n) can be viewed as a set of m environments e_1, \dots, e_m , n systems $\Pi_{k,j}$ of order q , and a set Φ of flags, in such a way that: (a) the links between the m environments are given by the arcs from the directed graph G ; (b) each environment has a flag from Φ at any instant; (c) all P systems have the same working alphabet, the same membrane structure and the same evolution rules; (d) each environment e_j contains several P systems, $\Pi_{1,j}, \dots, \Pi_{n_j,j}$, where each evolution rule has associated a computable function p_j , and each one of them has an initial multiset which depends on j . Furthermore, inside the environments, only objects from the alphabet Σ can exist; that is, there are symbols from the working alphabet that cannot circulate through the environments.

A *configuration* of the system at any instant t is a tuple whose components are the following: (a) the flags associated with each environment at instant t (initially f_1, \dots, f_m); (b) the multisets of objects present in the m environments at instant t (initially E_1, \dots, E_m); and (c) the multisets of objects associated with each of the regions of each P system $\Pi_{k,j}$ (initially $\mathcal{M}_{1,j}^k, \dots, \mathcal{M}_{q,j}^k$), together with the polarizations of their membranes (initially all membranes have a neutral polarization).

We assume that a global clock exists, marking the time for the whole system, that is, all membranes and the application of all rules (both from \mathcal{R}_E and \mathcal{R}) are synchronized in all environments.

The P system can pass from one configuration to another by using the rules from $\mathcal{R} = \mathcal{R}_E \cup \bigcup_{j=1}^m \mathcal{R}_j^k$ as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, and all applicable rules are simultaneously applied.

A rule of the type $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration membrane i has polarization α , contains multiset v and its parent (the environment if the membrane is the skin membrane) contains multiset u . When that rule is applied, multisets u, v produce u', v' , respectively, and the new polarization is α' (the value of function p in that moment provide the affinity of the application of that rule). For each j ($1 \leq j \leq m$) there is just one further restriction, concerning the consistency of charges: in order to apply several rules of \mathcal{R}_j^k simultaneously to the same membrane, all the rules must have the same electrical charge on their right-hand side.

A rule of the environment of the type $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j contains object x . When that rule is applied, object x passes from e_j to e_{j_1}, \dots, e_{j_h} possibly transformed into objects y_1, \dots, y_h , respectively (the value of function p_1 in that moment provide the affinity of the application of that rule).

A rule of the environment of the type $(\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j'}}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j contains the P system $\Pi_{k,j}$. When that rule is applied, the system $\Pi_{k,j}$ passes from environment e_j to environment $e_{j'}$ (the value of function p_2 in that moment provide the affinity of the application of that rule).

A rule of the environment of the type $\{f\}(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}}$, where e_{j_1} can be equal to e_j or not, is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j has flag f and contains the multiset u . When that rule is applied multiset u produces multiset v and environment e_j keep the same flag. This kind of rule can be applied many times in a computation step. The value of function p_3 in that moment provide the affinity of the application of that rule.

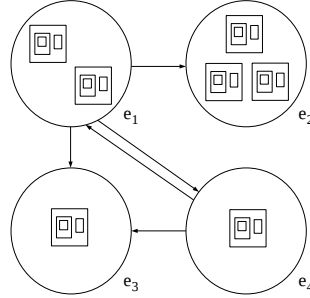
A rule of the environment of the type $\{f\}(u, f)_{e_j} \xrightarrow{p_4} (v, g)_{e_j}$ is applicable to a configuration at any instant t if the following is satisfied: in that configuration environment e_j has flag f and contains the multiset u . When that rule is applied multiset u produces multiset v and flag f of environment e_j is replaced by flag g . Bearing in mind that each environment only has a flag in any instant, this kind of rules can only be applied once in any moment.

Next, we depict the two approaches (stochastic and probabilistic) for multienvironment P systems.

3.1 Stochastic approach

We say that a multienvironment P system has a stochastic approach if the following holds:

- (a) The alphabet of flags, Φ , is an empty set.
- (b) The computable functions associated with the rules of the P systems are **propensities** (obtained from the kinetic constants) [22]: These rules is function of the time but they do not depend on the environment.
- (c) Initially, the P systems $\Pi_{k,j}$ are randomly distributed among the m environments of the system.



Multicompartmental P systems Multicompartmental P systems are multienvironment P systems with a stochastic approach which can be formally expressed as follows:

$$\Pi = (G, \Gamma, \Sigma, T, n = \sum_{j=1}^m n_j, \{\Pi_{k,j} \mid 1 \leq k \leq n_j, 1 \leq j \leq m\}, \{E_j \mid 1 \leq j \leq m\}, \mathcal{R}_E)$$

These systems can be viewed as a set of m environment connected by the arcs of a directed graph G . Each environment e_j only can contains P systems of the type $\Pi_{k,j}$. The total number of P systems is n , all of them with the same skeleton. The functions associated with the rules of the system are propensities which are computed as follows: stochastic constants are computed from kinetic constants by applying the mass action law, and the propensities are obtained from the stochastic constants by using the concentration of the objects in the LHS at any instant. In these systems there are rules of the following types:

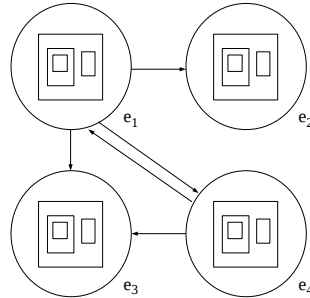
1. $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$
2. $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$
3. $(\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j'}}$

The dynamics of these systems is captured by the multicompartmental Gillespie's algorithm [22] or the deterministic waiting time [4]. Next, some practical examples of multicompartmental P systems applications are highlighted: Quorum sensing in *Vibrio Fischeri* [24], gene expression control in Lac Operon [25], and FAS-induced apoptosis [4]. A software environment supporting this model is Infobiotics Workbench [1], which provides (in version 0.0.1): a modelling language, a multi-compartmental stochastic simulator based on Gillespie's Stochastic Simulation Algorithm, a formal model analysis, and a structural and parameter model optimisation.

3.2 Probabilistic approach

We say that a multienvironment P system has a stochastic approach if the following holds:

- (a) The total number of P systems $\Pi_{k,j}$ is, at most, the number m of environment, that is, $n \leq m$.
- (b) Functions p_r associated with rule $r \equiv u[v]_i^\alpha \xrightarrow{p_r} u'[v']_i^{\alpha'}$ from $\Pi_{k,j}$ are **probability functions** such that for each $u, v \in M(\Gamma)$, $i \in \{1, \dots, q\}$, $\alpha \in \{0, +, -\}$, if r_1, \dots, r_z are the rules in R_j^k whose LHS is $u [v]_i^\alpha$, then
$$\sum_{j=1}^z p_{r_j}(t) = 1, \text{ for each } t (1 \leq t \leq T).$$
- (c) Functions p_1 associated with the rules of the environment $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ are **probability functions** such that for each $x \in \Sigma$ and each environment e_j , the sum of all functions associated with the rules whose LHS is $(x)_{e_j}$, is equal to 1.
- (d) Functions p_2 associated with the rules of the environment $(\Pi_{k,j})_{e_j} \xrightarrow{p_2} (\Pi_{k,j})_{e_{j'}}$ are constant functions equal to 0; that is, these rules will never be applied.
- (e) Functions p_3 associated with the rules of the environment $\{f\}(u)_{e_j} \xrightarrow{p_3} (v)_{e_{j_1}}$ are **probability functions**.
- (f) Functions p_4 associated with the rules of the environment $\{f\}(u, f)_{e_j} \xrightarrow{p_4} (v, g)_{e_j}$ are constant functions equal to 1.
- (g) There exist no rules $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$ in the skin membrane of $\Pi_{k,j}$ and rules of the environment $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ such that $x \in u$.
- (h) Initially, each environment e_j contains at most one P system $\Pi_{k,j}$.



Population Dynamics P systems (PDP) Population Dynamics P systems are multienvironment P systems with a probabilistic approach such that the alphabet Φ of the flags is an empty set and $n = m$, that is, the environment has not any flag and the total number n of P systems are equal to the number m of environments. Then in a PDP system Π each environment e_j contains

exactly one P system $\Pi_{k,j}$, which will be denoted henceforth by Π_j ; that is, $\forall j, 1 \leq j \leq m, n_j = 1$.

$$\Pi = (G, \Gamma, \Sigma, T, n, \{\Pi_j \mid 1 \leq j \leq m\}, \{E_j \mid 1 \leq j \leq m\}, \mathcal{R}_E)$$

In these systems there are rules of the following types:

1. $u[v]_i^\alpha \xrightarrow{p} u'[v']_i^{\alpha'}$
2. $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$

Let us recall that in this kind of systems each rule has an associated probability function that depends on the time and on the environment where the rule is applied.

Finally, in order to ease the understandability of the whole framework, Figure 1 shows a graphical summary of multienvironment P systems and the two approaches (stochastic and probabilistic).

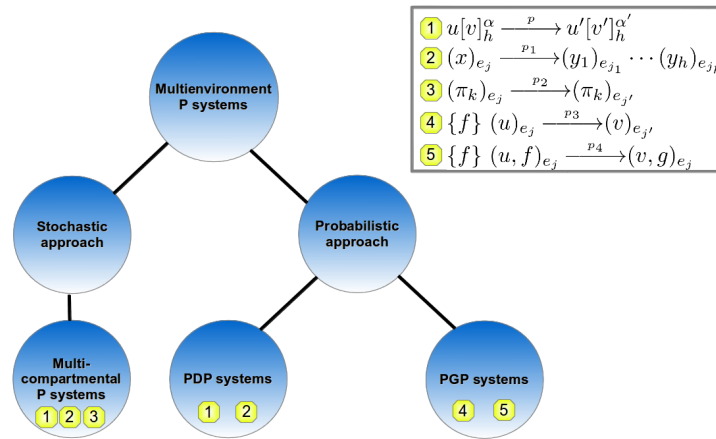


Fig. 1: The formal framework of Multienvironment P systems

Some practical examples of using PDP systems on the modelling of real ecosystems are: the Bearded Vulture at the Pyrenees (Spain) [6,3], the Zebra mussel at the Ribarroja reservoir (Spain) [3], and the Pyrenean Chamois [5]. A simple example of modelling pandemics dynamics can be seen in [2].

The dynamics of these systems is captured by the Direct Non-deterministic Distribution algorithm with Probabilities (DNDDP) algorithm [17], or the Direct distribution based on Consistent Blocks Algorithm (DCBA) [16]. DNDDP aims to perform a random distribution of rule applications without using the concept of rule block, but this selection process is biased towards those rules with

the highest probabilities. DCBA was first conceived to overcome the accuracy problem of DNDP, by performing an object distribution along the rule blocks, before applying the random distribution process. Although the accuracy achieved by the DCBA is better than the DNDP algorithm, the latter is much faster. In order to improve the performance of simulators implementing DCBA, parallel architectures has been used [15]. For example, a GPU-based simulator, using CUDA, reaches the acceleration of up to 7x, running on a NVIDIA Tesla C1060 GPU (240 processing cores). However, these accelerated simulators are still to be connected to those general environments to run virtual experiments. Therefore, P-Lingua and pLinguaCore are being utilised to simulate PDP systems [2,11]. The provided virtual experimentation environment is called MeCoSim [21], and it is based on P-Lingua.

4 Probabilistic Guarded P systems (PGP)

Probabilistic Guarded P systems are multienvironment P systems with a probabilistic approach such that $n = 0$, that is, there is no P systems $\Pi_{k,j}$ (so the alphabet Γ can be considered as an emptyset), and the alphabet of the flags, Φ , is a nonempty set such that every environment has a unique flag in the initial configuration.

Definition 2. A Probabilistic Guarded P system (PGP system, for short) of degree $m \geq 1$ is a tuple $\Pi = (G, \Sigma, \Phi, T, \{A_j \mid 1 \leq j \leq m\}, \mathcal{R}_E)$, where:

- $G = (V, S)$ is a directed graph whose set of nodes is $V = \{e_1, \dots, e_m\}$.
- Σ and Φ are finite alphabets such that $\Sigma \cap \Phi = \emptyset$. Elements in Σ are called **objects** and elements in Φ are called **flags**.
- For each j , $1 \leq j \leq m$, $A_j = E_j \cup \{f_j\}$, with $f_j \in \Phi$ and $E_j \in M(\Sigma)$. Thus, from now on, we “represent” A_j by the pair (f_j, E_j) .
- \mathcal{R}_E is a finite set of rules of the following types:
 - $\{f\}(u)_{e_j} \rightarrow (v)_{e_{j_1}}$ with $u, v \in M(\Sigma)$, $f \in \Phi$ and $1 \leq j, j_1 \leq m$.
 - $\{f\}(u, f)_{e_j} \rightarrow (v, g)_{e_j}$ with $u, v \in M(\Sigma)$, $f, g \in \Phi$ and $1 \leq j \leq m$.

There are no rules of types $\{f\}(u, f)_{e_j} \rightarrow (v, g)_{e_j}$ and $\{f\}(u)_{e_j} \rightarrow (v)_{e_{j_1}}$, for $f \in \Phi$, $1 \leq j, j_1 \leq m$ and $u \in M(\Sigma)$.

For each $f \in \Phi$ and $j, 1 \leq j \leq m$, there exists only one rule of type $\{f\}(u, f)_{e_j} \rightarrow (v, g)_{e_j}$.
- The arcs of graph $G = (V, S)$ are defined from \mathcal{R}_E as follows: $(e_j, e_{j_1}) \in S$ if and only if there exists a rule of the type $\{f\}(u)_{e_j} \rightarrow (v)_{e_{j_1}}$, or $j = j_1$ and there exists a rule of the type $\{f\}(u, f)_{e_j} \rightarrow (v, g)_{e_j}$.
- Each rule from \mathcal{R}_E has an associated probability, that is, there exists a function $p_{\mathcal{R}_E}$ from \mathcal{R}_E into $[0, 1]$, such that:
 - For each $f \in \Phi, u \in M(\Sigma), 1 \leq j \leq m$, if r_1, \dots, r_t are rules of the type $\{f\}(u)_{e_j} \rightarrow (v)_{e_{j_1}}$, then $\sum_{s=1}^t p_{\mathcal{R}_E}(r_s) = 1$.
 - If $r \equiv \{f\}(u, f)_{e_j} \rightarrow (v, g)_{e_j}$, then $p_{\mathcal{R}_E}(r) = 1$.

A Probabilistic Guarded P system can be viewed as a set of m environments, called *cells*, labelled by $1, \dots, m$ such that: (a) E_1, \dots, E_m are finite multisets over Σ representing the objects initially placed in the cells of the system; (b) f_1, \dots, f_m are flags that initially mark the cells; (c) G is a directed graph whose arcs specify connections among cells; (d) \mathcal{R}_E is the set of rules that allow the evolution of the system and each rule r is associated with a real number $p_{\mathcal{R}_E}(r)$ in $[0, 1]$ describing the probability of that rule to be applied in the case that it is applicable. That is to say, each rule r has a probability $p_{\mathcal{R}_E}(r)$ to be applied at an instant t for each possible application of r at such an instant. Γ and n are omitted, as they are $\Gamma = \emptyset$ and $n = 0$, respectively.

In PGP systems, two types of symbols are used: *objects* (elements in Σ) and *flags* (elements in Φ). It can be considered that objects are **in** cells and flags are **on** (the borderline of) cells.

4.1 Semantics of PGP systems

Definition 3. A configuration at any instant $t \geq 0$ of a PGP system Π is a tuple $C_t = (x_1, u_1, \dots, x_m, u_m)$ where, for each i , $1 \leq i \leq m$, $x_i \in \Phi$ and $u_i \in M(\Sigma)$. That is to say, such a configuration is described by all multisets of objects over Σ associated with all the cells present in the system and the flags marking these cells. $(f_1, E_1, \dots, f_m, E_m)$ is said to be the initial configuration of Π . At any instant, each cell has exactly one flag, in a similar manner to polarizations in cell-like P systems.

Definition 4. A rule r of the type $\{f\}(u)_i \rightarrow (v)_j$ is applicable to a configuration $C_t = (x_1, u_1, \dots, x_m, u_m)$ if and only if $x_i = f$ and $u \subseteq u_i$, for all $1 \leq i \leq m$.

When applying r to C_t , objects in u are removed from cell i and objects in v are produced in cell j . Flag f is not changed; it plays the role of a catalyst assisting the evolution of objects in u .

Definition 5. A rule r of the type $\{f\}(u, f)_i \rightarrow (v, g)_i$ $1 \leq i \leq m$ is applicable to a configuration $C_t = (x_1, u_1, \dots, x_m, u_m)$ if and only if $x_i = f$ and $u \subseteq u_i$.

When applying r to C_t , in cell i objects in u are replaced by those in v and f is replaced by g . In this case, flag f is consumed, so r can be applied only once in instant t in cell i .

Remark 1. After applying a rule r of the type $\{f\}(u, f)_i \rightarrow (v, g)_i$, other rules r' of the type $\{f\}(u)_i \rightarrow (v)_j$ can still be applied (the flag remains in vigor). However, f has been consumed (in the same sense that an object $x \in \Sigma$ is consumed), so no more rules of the type $\{f\}(u, f)_i \rightarrow (v, g)_i$ can be applied.

Definition 6. A configuration is a halting configuration if no rule is applicable to it.

Definition 7. We say that configuration C_1 yields configuration C_2 in a transition step if we can pass from C_1 to C_2 by applying rules from \mathcal{R}_ε in a non-deterministic, maximally parallel manner, according to their associated probabilities denoted by map $p_{\mathcal{R}_\varepsilon}$. That is to say, a maximal multiset of rules from \mathcal{R}_ε is applied, no further rule can be added.

Definition 8. A computation of a PGP system Π is a sequence of configurations such that: (a) the first term of the sequence is the initial configuration of Π , (b) each remaining term in the sequence is obtained from the previous one by applying the rules of the system following Definition 7, (c) the sequence will have, at most, $T + 1$ terms. That is, we consider that the execution of a PGP system will perform, at most, T steps.

4.2 Comparison between PGP systems and other frameworks in Membrane Computing

Probabilistic Guarded P systems (*PGP systems*) display similarities with other frameworks in Membrane Computing. For example, *P systems with proteins on membranes* [18] are a type of cell-like systems in which membranes might have attached a set of proteins which regulate the application of rules, whilst in PGP systems each cell has only one flag. Therefore, some rules are applicable if and only if the corresponding protein is present.

When comparing PGP systems and *Population Dynamics P systems* [2], it is important to remark the semantic similarity between flags and polarizations, as they both define at some point the context of each compartment. Nevertheless, as described at the beginning of this paper, upon the application of a rule $r \equiv \{f\} (u, f)_i \rightarrow (v, g)_i$ flag f is consumed, thus ensuring that r can be applied at most once to any configuration. This property keeps PGP transitions from yielding inconsistent flags; at any instant, only one rule at most can change the flag in each membrane, so scenarios in which inconsistent flags produced by multiple rules are impossible. Moreover, in PDP systems the number of polarizations is limited to three (+, - and 0), whereas in their PGP counterpart depends on the system itself. Finally, each compartment in PDP systems contains a hierarchical structure of membranes, which is absent in PGP systems.

5 Simulation of PGP systems

When simulating PGP systems, there exist two cases, according to if there exists object competition or not. In this work, only algorithms for the second case are introduced, but some ideas are given to handle object competition among rules in the model, and kept for future developments.

5.1 Some definitions on the model

The following concepts defined for PGP systems are analogous to those described in [16], but adapted to the syntax of PGP systems.

Remark 2. For the sake of simplicity, henceforth the following notation will be used. For every cell i , $1 \leq i \leq m$, and time t , $0 \leq t \leq T$, the flag and multiset of cell i in step t are denoted as $x_{i,t} \in \Phi$ and $u_{i,t} \in M(\Sigma)$, respectively. Similarly, $|u|_y$, where $u \in M(\Sigma)$, $y \in \Sigma$ denotes the number of objects y in multiset u .

Definition 9 shows the notation regarding the left-hand and right-hand sides of rules.

Definition 9. For each rule $r \in \mathcal{R}_{\mathcal{E}}$:

Type 1: If r is of the form $r \equiv \{f\}(u)_i \rightarrow (v)_j$, we denote the left-hand side of r ($LHS(r)$) as $LHS(r) = (i, f, u)$ and the right-hand side of r ($RHS(r)$) as $RHS(r) = (j, f, v)$.

Type 2: If r is of the form $r \equiv \{f\}(u, f)_i \rightarrow (v, g)_i$, we denote the left-hand side as $LHS(r) = (i, f, u, f)$ and the right-hand side as $RHS(r) = (i, g, v)$.

Let us recall that for each i , $1 \leq i \leq m$ and $f \in \Phi$, there exists a *unique* rule of type 2: $r \equiv \{f\}(u, f)_i \rightarrow (v, g)_i$.

Next, Definition 10 introduces the concept of rule blocks in PGP systems, which is inspired by the one used in PDP systems [16].

Definition 10. For each $1 \leq i \leq q$, $f \in \Phi$, and $u \in M(\Sigma)$, we will denote:

- The block of communication rules $B_{i,f,u}^1 = \{r \in \mathcal{R} : LHS(r) = (i, f, u)\}$; that is, the set of rules of type 1 having the same left-hand side.
- The block of context-changing rules $B_{i,f,u}^2 = \{r \in \mathcal{R} : LHS(r) = (i, f, u, f)\}$; that is, the set of rules of type 2 having the same left-hand side.

$B_{i,f,u}^1 \cap B_{i,f,u}^2 = \emptyset$. It is important to recall that, as it is the case in PDP systems, the sum of probabilities of all the rules belonging to the same block is always equal to 1 – in particular, rules with probability equal to 1 form individual blocks. Consequently, non-empty blocks of context-changing rules (type 2) are composed of single rules. In addition, rules with overlapping (but different) left-hand sides are classified into different blocks.

Definition 11. For each i , $1 \leq i \leq m$, we will consider the set of all rule blocks associated with cell i as $B_i = \{B_{i,f,u}^1, B_{i,f,u}^2 : f \in \Phi \wedge u \in M(\Sigma)\}$.

We will also consider a total order in B_i , for $1 \leq i \leq m$, $B_i = \{B_{i,1}, B_{i,2}, \dots, B_{i,\alpha_i}\}$. Therefore, there are α_i blocks associated to cell i .

Furthermore, let $B_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq \alpha_i$ be a block associated to cell i . We define the following notations:

- $Type(B_{i,j})$ is equal to:
 - 1, if $\exists f \in \Phi, u \in M(\Sigma)$ such that $B_{i,j} = B_{i,f,u}^1$
 - 2, if $\exists f \in \Phi, u \in M(\Sigma)$ such that $B_{i,j} = B_{i,f,u}^2$
- $Flag(B_{i,j}) = f$, if $\exists k(1 \leq k \leq 2) \wedge \exists u \in M(\Sigma)$ such that $B_{i,j} = B_{i,f,u}^k$
- $Mult(B_{i,j}) = u$, if $\exists k(1 \leq k \leq 2) \wedge \exists f \in \Phi$ such that $B_{i,j} = B_{i,f,u}^k$

In addition, for each block $B_{i,j}$, $1 \leq i \leq m$ and $1 \leq j \leq \alpha_i$, associated to cell i , we consider a total order in its set of rules: $B_{i,j} = \{r_{i,j,1}, \dots, r_{i,j,h_{i,j}}\}$, where $h_{i,j}$ ($1 \leq i \leq m$, $1 \leq j \leq \alpha_i$) denotes the number of rules in block $B_{i,j}$. Obviously, all the rules of a block are of the same type.

Definition 12. A PGP system is said to feature object competition, if there exists at least two different blocks $B_{i,j}$ and $B_{i,j'}$ (possibly of different type), such that $Flag(B_{i,j}) = Flag(B_{i,j'})$, and $Mult(B_{i,j}) \cap Mult(B_{i,j'}) \neq \emptyset$. That is, their rules have overlapping (but not equal) left-hand sides.

Remark 3. It is worth noting that all rules in the model can be consistently applied. This is because there can only exist one flag $f \in \Phi$ at every membrane at the same time, and, consequently, at most one context-changing rule $r \equiv \{f\}(u, f)_i \rightarrow (v, g)_i$ can consume f and replace it (where possibly $f = g$).

Definition 13. Given a block $B_{i,f,u}^1$ or $B_{i,f,u}^2$, where $u \in M(\Sigma)$, $f \in \Phi$, $1 \leq i \leq m$ and a configuration $C_t = \{x_1, u_1, \dots, x_m, u_m\}$, $0 \leq t \leq T$, the maximum number of applications of such a block in C_t is the maximum times that each one of its rules can be applied in C_t .

5.2 Simulation Algorithm

Next, we define some auxiliary data structures to be used in the simulation algorithms.

NBA (Number of Block Applications): a matrix of integer numbers of dimension $m \times N_{BM}$, where $N_{BM} = \max(\alpha_i \mid 1 \leq i \leq m)$ (maximum number of blocks for all cells). Each element $NBA_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq N_{BM}$ stores the number of applications of block $B_{i,j}$.

NRA (Number of Rule Applications): a matrix of integer numbers of dimension $m \times N_{BM} \times N_{RM}$, where $N_{RM} = \max(h_{i,j})$, $1 \leq i \leq m$, $1 \leq j \leq \alpha_i$ (maximum number of rules for all blocks in all membranes). Each element $NRA_{i,j,k}$, $1 \leq i \leq m$, $1 \leq j \leq \alpha_i$, $1 \leq k \leq h_{i,j}$, stores the number of applications of rule $r_{i,j,k}$, identified by its cell, block and local identifier inside its block, according to the established total order.

The algorithm for simulation of PGP systems receives three parameters:

- The PGP system Π of degree m .
- The integer number $T > 0$ (number of time steps).
- An integer number $K > 0$ (random accuracy). It indicates for how many cycles block applications are assigned among their rules in random fashion. That is, the algorithm distributes the applications of each block among its rules for K cycles, and after that, block applications are maximally assigned among rules in a single cycle. Therefore, the greater K is, the more accurate the distribution of rule applications for each block becomes, but at the expense of a greater computational cost. It is used as an accuracy parameter for the probabilistic method. Algorithm 5.4 performs this function.

When simulating PGP systems without object competition, it is not necessary to randomly assign objects among blocks; as they do not compete for objects, then the number of times that each block is applied is always equal to its maximum number of applications. As it is the case of DCBA for PDP systems [16], the simulation algorithm heavily relies on the concept of block, being rule applications secondary. However, DCBA handles object competition among blocks, penalizing more those blocks which require a larger number of copies of the same object which is inspired by the amount of energy required to join individuals from the same species. On the other hand, object competition is not supported on the proposed algorithm. Algorithm 5.1 describes a simulation algorithm for PGP systems without object competition.

Algorithm 5.1 Algorithm for simulation of PGP systems

Input:

- T : an integer number $T \geq 1$ representing the iterations of the simulation.
- K : an integer number $K \geq 1$ representing non-maximal rule iterations (i.e., iterations in which the applications selected for each rule do not necessarily need to be maximal).
- $\Pi = (G, \Sigma, \Phi, T, \{(f_j, E_j) \mid 1 \leq j \leq m\}, \mathcal{R}_E)$: a PGP system of degree $m \geq 1$.

```

1: Initialization ( $\Pi$ )
2: for  $t \leftarrow 1$  to  $T$  do                                     ▷ See Algorithm 5.2
3:    $C'_t \leftarrow C_{t-1}$ 
4:   SELECTION of rules:
5:     PHASE 1: Objects distribution ( $C'_t$ )                       ▷ See Algorithm 5.3
6:     PHASE 2: Rule application distribution ( $C'_t$ )             ▷ See Algorithm 5.4
7:   EXECUTION of rules:
8:     PHASE 3: Object production ( $C'_t$ )                         ▷ See Algorithm 5.5
9:    $C_t \leftarrow C'_t$ 
10: end for

```

On each simulation step t , $1 \leq t \leq T$ and cell i , $1 \leq i \leq m$, the following stages are applied: *Object distribution* (selection), *Rule application distribution* (selection) and *Object generation* (execution).

However, before starting the simulation process, we must initialize some data structures. In *Initialization* (Algorithm 5.2), the initial configuration C_0 is constructed with the input PGP system Π . Moreover, the information about blocks are created; that is, the blocks of rules are computed, and ordered for each cell. Moreover, the rules inside each block are also ordered. Finally, the data structures *NBA* and *NRA* are initialized with zeros.

In the *Object distribution* stage (Algorithm 5.3), objects are distributed among blocks. As the system to simulate does not feature object competition, the number of applications of each block is its maximum. Then, objects are consumed accordingly. It is in this stage that the flag checking for each block

Algorithm 5.2 Initialization**Input:** $\Pi = (G, \Sigma, \Phi, T, \{(f_j, E_j) \mid 1 \leq j \leq m\}, \mathcal{R}_E)$

```

1:  $C_0 \leftarrow \{f_1, E_1, \dots, f_m, E_m\}$  ▷ Initial configuration
2: for  $i \leftarrow 1$  to  $m$  do ▷ For each cell
3:    $B_i \leftarrow$  ordered set of blocks formed by rules of  $\mathcal{R}$  associated with cell  $i$ 
4:    $\alpha_i \leftarrow |B_i|$  ▷ Number of rule blocks
5:   for  $j \leftarrow 1$  to  $\alpha_i$  do ▷ For each block associated with the cell
6:      $B_{i,j} \leftarrow$  ordered set of rules from  $j^{\text{th}}$  block in  $B_i$ .
7:      $h_{i,j} \leftarrow |B_{i,j}|$  ▷ Number of rules within the block
8:      $NBA_{i,j} \leftarrow 0$  ▷ Initially, all blocks applications are 0
9:     for  $k \leftarrow 1$  to  $h_{i,j}$  do ▷ Initially, all rule applications are 0
10:       $NRA_{i,j,k} \leftarrow 0$ 
11:     end for
12:   end for
13: end for

```

is performed. Moreover, blocks of type 2 (context-changing rules) consume and generate the new flag.

Algorithm 5.3 Phase 1: Object distribution among blocks**Input:** $C'_t = \{x_{1,t}, u_{1,t}, \dots, x_{m,t}, u_{m,t}\}$

```

1: for  $i \leftarrow 1$  to  $m$  do ▷ For each cell
2:   for  $j \leftarrow 1$  to  $\alpha_i$  do ▷ For each block associated with the cell
3:     if  $Flag(B_{i,j}) = x_{i,t}$  then
4:       if  $Type(B_{i,j}) = 1 \wedge Mult(B_{i,j}) \subseteq u_{i,t}$  then
5:          $NBA_{i,j} \leftarrow \min(\lfloor \frac{|u_{i,t}|_z}{|Mult(B_{i,j})|_z} \rfloor : z \in \Sigma)$  ▷ Maximal application
6:          $u_{i,t} \leftarrow u_{i,t} - NBA_{i,j} \cdot Mult(B_{i,j})$  ▷ Update the configuration
7:       end if
8:       if  $Type(B_{i,j}) = 2 \wedge Mult(B_{i,j}) \subseteq u_{i,t}$  then
9:          $NBA_{i,j} \leftarrow 1$  ▷ Just one application
10:         $x_{i,t} \leftarrow g$ , being  $RHS(r_{i,j,1}) = (i, g, v)$  with  $B_{i,j} = \{r_{i,j,1}\}$  ▷ Update cell flag
11:         $u_{i,t} \leftarrow u_{i,t} - NBA_{i,j} \cdot Mult(B_{i,j})$  ▷ Update the configuration
12:       end if
13:     end if
14:   end for
15: end for

```

Next, objects are distributed among rules according to a binomial distribution with rule probabilities and maximum number of block applications as parameters. This algorithm is composed of two stages *non-maximal* and *maximal* repartition. In the non-maximal repartition stage, a rule in the block is randomly selected according to a uniform distribution, so each rule has the same probability to be chosen. Then, its number of applications is calculated according to

an *ad-hoc* procedure based on a binomially distributed variable $Binomial(n, p)$, where n is the remaining number of block applications to be assigned among its rules and p is the corresponding rule probability. This process is repeated a number K of iterations for each block $B_{i,j}$, $1 \leq i \leq m, 1 \leq j \leq \alpha_i$. We propose this procedure to simulate a multinomial distribution, but it could be easily interchangeable by another one. Algorithm 5.4 describes this procedure. If, after this process, there are still applications to assign among rules, a rule per applicable block is chosen at random and as many applications as possible are assigned to it in the maximal repartition stage. An alternative approach would be to implement a multinomial distribution of applications for the rules inside each block, such as the way that it is implemented on the DCBA algorithm [16]. A method to implement a multinomial distribution would be the conditional distribution method, which emulates a multinomial distribution based on a sequence of binomial distributions [9]. This would require to normalize rule probabilities for each rule application distribution iteration. This approach has also been tested on the simulation algorithm, but was discarded because it tends to distribute too few applications in the non-maximal repartition stage, thus leaving too many applications for the rule selected in the maximal repartition one.

Algorithm 5.4 Phase 2: Rule application distribution

Input: $C'_t = \{x_{1,t}, u_{1,t}, \dots, x_{m,t}, u_{m,t}\}$

```

for  $k \leftarrow 1$  to  $K$  do
    for  $i \leftarrow 1$  to  $m$  do
        for  $j \leftarrow 1$  to  $\alpha_i$  do
             $l \leftarrow Uniform\{1, \dots, h_{i,j}\}$ 
             $lnrap \leftarrow Binomial(NBA_{i,j}, p_{\mathcal{R}}(r_{i,j,l}))$ 
             $NRA_{i,j,l} \leftarrow NRA_{i,j,l} + lnrap$ 
             $NBA_{i,j} \leftarrow NBA_{i,j} - lnrap$ 
        end for
    end for
end for
for  $i \leftarrow 1$  to  $m$  do
    for  $j \leftarrow 1$  to  $\alpha_i$  do
         $l \leftarrow Uniform\{1, \dots, h_{i,j}\}$ 
         $NRA_{i,j,l} \leftarrow NRA_{i,j,l} + NBA_{i,j}$ 
         $NBA_{i,j} \leftarrow 0$ 
    end for
end for

```

▷ Non-maximal repartition stage

▷ Select a random rule $r_{i,j,l}$ in Block $B_{i,j}$

▷ Update rule applications

▷ Maximal repartition stage

Lastly, rules produce objects as indicated by their right-hand side. Each rule produces objects according to its previously assigned number of applications. Algorithm 5.5 describes this procedure.

The algorithm proposed in this paper works only for models without object competition. This is because the models studied so far (such as the *Pieris oleracea* model to be mentioned on Section 6) did not have object competition,

Algorithm 5.5 Phase 3: Object production

```

for  $i \leftarrow 1$  to  $m$  do                                     ▷ For each cell
  for  $j \leftarrow 1$  to  $\alpha_i$  do                               ▷ For each block associated with the cell
    for  $k \leftarrow 1$  to  $h_{i,j}$  do                               ▷ For each rule belonging to the block
       $u_{i,t} \leftarrow u_{i,t} + NRA_{i,j,k} \cdot v$ , where  $RHS(r_{i,j,k}) = (i', f', v)$ 
       $NRA_{i,j,k} \leftarrow 0$ 
    end for
  end for
end for

```

so this feature was not required. However, it might be interesting to develop new algorithms supporting it. They would be identical to their counterpart without object competition, solely differing in the protocol by which objects are distributed among blocks. As an example, it would be possible to adapt the way in which objects are distributed in the DCBA algorithm [16].

5.3 Software environment

This section provides an overview of the developed simulators, the P-Lingua extension, and the GUI for PGP systems.

Simulators A simulator for PGP systems without object competition has been incorporated on P-Lingua [11]. In addition, a C++ simulator for PGP systems (namely PGPC++) has also been implemented. The libraries used for random number generation are COLT [26] in the P-Lingua simulator, and standard `std::rand` [27] for PGPC++. In the latter, the facilities provided by `std::rand` are directly used. These libraries provide a wide range of functionality to generate and handle random numbers, and are publicly available under open source licenses.

P-Lingua extension In order to define PGP systems, P-Lingua has been extended to support PGP rules. Specifically, given $f, g \in \Phi$, $u, v \in M(\Sigma)$, $1 \leq i, j \leq m$, $p = p_{\mathcal{R}}(r)$, rules are represented as follows:

$$\{f\}(u)_i \xrightarrow{p} (v)_j, \equiv \text{@guard } f \text{ ?}[u]'i \text{ --> } [v]'j \text{ :: } p;$$

$$\{f\}(u, f)_i \rightarrow (v, g)_i \equiv \text{@guard } f \text{ ?}[u, f]'i \text{ --> } [v, g]'i \text{ :: } 1.0;$$

In both cases, if $p = 1.0$, then `:: p` can be omitted. If $i = j$, then $\{f\}(u)_i \xrightarrow{p} (v)_j$ can be written as `@guard f ?[u --> v]'i :: p`; . Likewise, $\{f\}(u, f)_i \rightarrow (v, g)_i$ can always be written as `@guard f ?[u, f --> v, g]'i`;

Further additional constructs have been included to ease parametrization of P systems. The idea is to enable completely parametric designs, so as experiments can be tuned by simply adjusting parameters leaving modifications of P-Lingua

files for cases in which changes in semantics are in order. This extension is going to be released on the next version of P-Lingua.

In addition, two new formats have been integrated into P-Lingua. These formats (XML-based and binary) encode P systems representing labels and objects as numbers instead of strings, so they are easily parsed and simulated by third-part simulators such as PGPC++.

A graphical environment for PGP systems *MeCoGUI* is a new Graphical User Interface (GUI) developed for the simulation of PGP systems. MeCoSim [21] could have been used instead. However, in the environment in which the simulators were developed there exist some pros and cons on this approach versus an ad-hoc simulator.

MeCoSim is an integrated development environment (IDE). That is to say, it provides all functionality required for the simulation and computational analysis of P systems. To define the desired input and output displays, it is necessary to configure a spreadsheet by using an *ad-hoc* programming language. However, it would entail teaching this language to prospective users, who can be proficient in any other statistic programming language instead, such as R. In this sense, a more natural approach for them is to develop a GUI in which users can define input parameters and results analysis on R.

To do so, the developed GUI takes as input a P system file on P-Lingua format and a CSV file encoding its parameters, and outputs a CSV file which contains simulation results. This way, users can define inputs and analyse outputs on their programming language of choice. CSV is a widespread, simple and free format with plenty of libraries for different languages. This flexibility comes at the cost concerning that the developed GUI is not an IDE, as input parameters and simulation analysis cannot be directly input and viewed on the GUI. Rather, it is necessary to develop applications to generate and process these CSV files which depend on the domain of use. In some simulators (such as PGPC++), the output CSV files represent labels and objects as integers, but this application includes a button to translate output files from PGPC++ into string-representative file formats. Figure 2 displays the main screen of this application.

MeCoGUI can also translate P systems into machine-readable formats, such as those read by PGPC++. Finally, it is important to remark that these applications play the role of domain-specific spreadsheets on MeCoSim, so MeCoGUI can simulate any type of P system supported by P-Lingua. This is because only external applications for input data and simulation processing depend on the domain, not MeCoGUI itself, which is general for any type of P system.

6 Applications of PGP systems

A model of the ecosystem of the white cabbage butterfly (*Pieris oleracea*) [8], based on PGP systems, is a currently ongoing project. Such a species is suffering

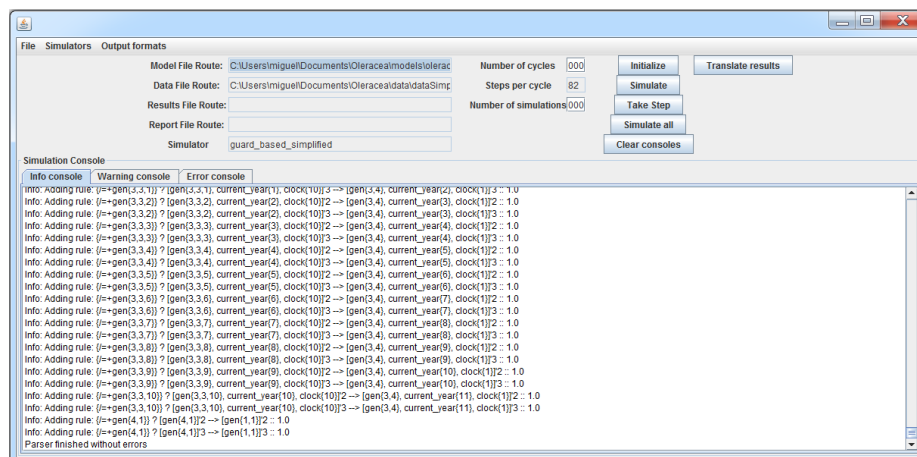


Fig. 2: Main screen of MeCoGUI

the invasion of the garlic mustard (*Alliaria petiolata*), which is replacing native host broadleaf toothwort (*Cardamine diphylla*) and ravaging the butterfly's natural habitat. Specifically, *A. petiolata* contains a deterrent agent for larvae of *P. oleracea*. Moreover, such a plant is toxic for these larvae, although it contains a chemical compound which lures mature butterflies and frames them into laying eggs. Nevertheless, a minority of individuals tolerates such a deterrent, metabolize the toxin and reach the pupa stage [13,14].

The distribution of phylogenetic profiles across the species consists of a majority of homozygous individuals unable to thrive on *A. petiolata* patches, a minority of homozygous individuals which do well on *A. petiolata* rosettes and, in the midterm, an slightly larger population of heterozygous individuals with both alleles. The allele which enables butterflies to overcome the dietary restrictions imposed by *A. petiolata* is dominant, but individuals carrying this allele undergo a detoxification mechanism which entails an energetic cost and hampers their arrival at adulthood [13].

The model under development aims to identify if there has been any evolutionary adaptation of the butterfly species significant enough so as to ensure its survival in the new scenario. Specifically, the idea is to assess if the detoxification cost associated with individuals tolerating *A. petiolata* pays off in the new scenario or, on the other hand, the phylogenetic distribution will stay the same and other mechanism will come into effect, such as hybridization with other butterfly species such as *Pieris rapae* [8].

The approach taken in this project aims to validate the model *qualitatively*. A *qualitative validation* is defined as follows: a model is qualitatively validated if it can reproduce some properties verified by the ecosystem under different scenarios (according to the experts).

7 Conclusions and Future Work

Multienvironment P systems are a general, formal framework for modelling population dynamics in biology. The framework has two main approaches: stochastic (micro-level oriented) and probabilistic (macro-level oriented). The framework has been extended in the probabilistic approach, with the inclusion of a new modelling framework called Probabilistic Guarded P (PGP) systems. PGP systems are inspired by Population Dynamics P systems, and aim to simplify the design and simulation of models of ecological phenomena. The model has been formalized in this paper, and a simulation algorithm is introduced. This algorithm is restricted for models which do not feature object competition. Moreover, an extension of the P-Lingua language is provided to enable PGP systems in P-Lingua, as well as a Graphical User Interface (GUI) to simulate PGP systems (MeCoGUI).

The framework of PGP systems is being utilised for modelling the ecosystem of *Pieris napi oleracea*, a butterfly native to Northeast U.S.A. The aim is to validate the model qualitatively; that is, checking that if the ecosystem verifies some properties under different scenarios (experts), our model reproduces those properties as well.

Although PGP systems provide a simplified alternative to PDP systems, some constraints to the supported models are imposed: only models without object competition are allowed. Therefore, future research lines will be focused on overcoming this constraint, providing new simulation algorithms permitting object competition. Moreover, new case studies will be considered, which can help to extend the framework. Finally, PGP simulation will be accelerated by using parallel architectures, such as GPU computing with CUDA.

Acknowledgements

The authors acknowledge the support of the project TIN2012-37434 of the “Ministerio de Economía y Competitividad” of Spain, co-financed by FEDER funds. Manuel García-Quismondo also acknowledges the support from the National FPU Grant Programme from the Spanish Ministry of Education. Miguel A. Martínez-del-Amor also acknowledges the support of the 3rd Postdoctoral phase of the PIF program associated with “Proyecto de Excelencia con Investigador de Reconocida Valía” of the “Junta de Andalucía” under grant P08-TIC04200.

References

1. J. Blakes, J. Twycross, F.J. Romero-Campero, N. Krasnogor. The Infobiotics Workbench: an integrated in silico modelling platform for Systems and Synthetic Biology, *Bioinformatics*, **27**, 23 (2011), 3323-3324.

2. M.A. Colomer, 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. *Applications of Membrane Computing in Systems and Synthetic Biology. Emergence, Complexity and Computation series*, Volume **7**. Chapter 4, pp. 97–132, 2014, Springer Int. Publishing.
3. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A computational modeling for real ecosystems based on P systems, *Natural Computing*, **10**, 1 (2011), 39–53.
4. S. Cheruku, A. Păun, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems, *Progress in Natural Science*, **17**, 4 (2007), 424–431.
5. M.A. Colomer, S. Lavín, I. Marco, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, E. Serrano, L. Valencia-Cabrera. Modeling population growth of Pyrenean Chamois (*Rupicapra p. pyrenaica*) by using P systems, *LNCS*, **6501** (2011), 144–159.
6. M.A. Colomer, A. Margalida, D. Sanuy, M.J. Pérez-Jiménez. A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study, *Ecological modelling*, **222**, 1 (2011), 33–47.
7. M.A. Colomer, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. A uniform framework for modeling based on P Systems. *Proceedings IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010)*, Volume **1**, pp. 616–621.
8. F.S. Chew. Coexistence and local extinction in two pierid butterflies, *The American Naturalist*, **118**, 5 (1981), 655–672.
9. C.S. Davis. The computer generation of multinomial random variates. *Computational Statistics and Data Analysis*, **16**, 2 (1993), 205–217.
10. P. Frisco, M. Gheorghe, M. J. Pérez-Jiménez (eds.) *Applications of Membrane Computing in Systems and Synthetic Biology*, Springer, 2014.
11. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Agustín Riscos-Núñez. An overview of P-Lingua 2.0, *LNCS*, **5957** (2010), 264–288.
12. M. Gheorghe, F. Ipaté, C. Dragomir, L. Mierla, L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez. Kernel P systems - Version I, *Proceedings of the Eleventh Brainstorming Week on Membrane Computing (BWMC2013)*, 2013, pp. 97–124.
13. M.S. Keeler, F.S. Chew. Escaping an evolutionary trap: preference and performance of a native insect on an exotic invasive host, *Oecologia*, **156**, 3 (2008), 559–568.
14. M.S. Keeler, F.S. Chew, B.C. Goodale, J.M. Reed. Modelling the impacts of two exotic invasive species on a native butterfly: top-down vs. bottom-up effects, *Journal of Animal Ecology*, **75**, 3 (2006), 777–788.
15. M.A. Martínez-del-Amor, I. Pérez-Hurtado, A. Gastalver-Rubio, A.C. Elster, M.J. Pérez-Jiménez. Population Dynamics P systems on CUDA. *LNBI*, **7605** (2012), 247–266.
16. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Romero-Jiménez, C. Graciani, A. Riscos-Núñez, M.A. Colomer, M.J. Pérez-Jiménez. DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution, *LNCS*, **7762** (2012), 27–56.

17. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, F. Sancho-Caparrini. A simulation algorithm for multienvironment probabilistic P systems: A formal verification, *International Journal of Foundations of Computer Science*, **22**, 1 (2011), 107–118.
18. A. Păun, B. Popa. P systems with proteins on membranes, *Fundamenta Informaticae*, **72**, 4 (2006), 467–483.
19. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and TUCS Report No 208.
20. G. Păun, G. Rozenberg, A. Salomaa (eds.). *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
21. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez. MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems, *Proceedings IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010)*, **volume I** (2010), pp. 637–643.
22. M.J. Pérez-Jiménez, F.J. Romero. P systems. A new computational modelling tool for Systems Biology. *Transactions on Computational Systems Biology VI. Lecture Notes in Bioinformatics*, **4220** (2006), 176–197.
23. L. Pan, M.J. Pérez-Jiménez. Computational complexity of tissue-like P systems, *Journal of Complexity*, **26**, 3 (2010), 296–315.
24. F.J. Romero-Campero, M.J. Pérez-Jiménez. A model of the Quorum Sensing system in *Vibrio fischeri* using P systems, *Artificial Life*, **14**, 1 (2008), 95–109.
25. F.J. Romero-Campero, M.J. Pérez-Jiménez. Modelling gene expression control using P systems: The Lac Operon, a case study, *BioSystems*, **91**, 3 (2008), 438–457.
26. *COLT library*. <http://acs.lbl.gov/software/colt/index.html>
27. *RAND function in C++/C Standard General Utilities Library (cstdlib)*. <http://www.cplusplus.com/reference/cstdlib/rand>

Solving the ST-Connectivity Problem with Pure Membrane Computing Techniques

Zsolt Gazdag¹ and Miguel A. Gutiérrez–Naranjo²

¹ Department of Algorithms and their Applications
Faculty of Informatics
Eötvös Loránd University, Budapest, Hungary
`gazdagzs@inf.elte.hu`

² Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
`magutier@us.es`

Abstract. In Membrane Computing, the solution of a decision problem X belonging to the complexity class \mathbf{P} via a polynomially uniform family of recognizer P systems is trivial, since the polynomial encoding of the input can involve the solution of the problem. The design of such solution has one membrane, two objects, two rules and one computation step. *Stricto sensu*, it is a solution in the framework of Membrane Computing, but it does not use Membrane Computing strategies. In this paper, we present three designs of uniform families of P systems that solve the decision problem STCON by using Membrane Computing strategies (*pure* Membrane Computing techniques): P systems with membrane creation, P systems with active membranes with dissolution and without polarizations and P systems with active membranes without dissolution and with polarizations. Since STCON is \mathbf{NL} -complete, such designs are *constructive* proofs of the inclusion of \mathbf{NL} in \mathbf{PMC}_{MC} , $\mathbf{PMC}_{AM^0_d}$ and $\mathbf{PMC}_{AM^+_d}$.

1 Introduction

Membrane Computing [13] is a well-established model of computation inspired by the structure and functioning of cells as living organisms able to process and generate information. It starts from the assumption that the processes taking place in the compartmental structures as living cells can be interpreted as computations. The devices of this model are called *P systems*.

Among the different research lines in Membrane Computing, one of the most vivid is the search of frontiers between complexity classes of decision problems, i.e., to identify collections of problems that can be solved (or languages that can be decided) by families of P systems with *similar* computational resources. In order to settle the correspondence between complexity classes and P system families, recognizer P systems were introduced in [9, 10]. Since then, recognizer P systems are the natural framework to study and solve decision problems within Membrane Computing.

In the last years, many papers have been published about the problem of deciding if a uniform family of recognizer P systems of type \mathcal{F} built in polynomial time is able to solve a given decision problem X . This is usually written as the problem of deciding if X belongs to $\mathbf{PMC}_{\mathcal{F}}$ or not. It has been studied for many P system models \mathcal{F} and for many decision problems X (see, e.g., [2–5] and references therein).

The solution of a decision problem X belonging to the complexity class \mathbf{P} via a polynomially uniform family of recognizer P systems is trivial (see [8, 11]), since the polynomial encoding of the input can involve the solution of the problem. On the one hand, by definition, $X \in \mathbf{P}$ if there exists a deterministic algorithm A working in polynomial time that *solves* X . On the other hand, the belonging of X to $\mathbf{PMC}_{\mathcal{F}}$ requires a polynomial time mapping *cod* that encodes the instances u of the problem X as multisets which will be provided as inputs. Formally, given a decision problem X and an algorithm A as described above, two different functions s (*size*) and *cod* (*encoding*) can be defined for each instance u of the decision problem:

- $s(u) = 1$, for all u
- $\text{cod}(u) = \begin{cases} \text{yes} & \text{if } A(u) = \text{yes} \\ \text{no} & \text{if } A(u) = \text{no}. \end{cases}$

The family of P systems which solves X is $\mathbf{II} = \{\Pi(n)\}_{n \in \mathbb{N}}$ with

$$\Pi(n) = \langle \Gamma, \Sigma, H, \mu, w, \mathcal{R}, i \rangle, \text{ where}$$

- *Alphabet*: $\Gamma = \{\text{yes}, \text{no}\}$
- *Input alphabet*: $\Sigma = \Gamma$
- *Set of labels*: $H = \{\text{skin}\}$
- *Membrane structure*: $[\]_{\text{skin}}$
- *Initial multisets*: $w = \emptyset$
- *Input label*: $i = \text{skin}$
- *Set of rules*: $[\text{yes}]_{\text{skin}} \rightarrow \text{yes} [\]_{\text{skin}}$ and $[\text{no}]_{\text{skin}} \rightarrow \text{no} [\]_{\text{skin}}$. Both are send-out rules.

Let us notice that \mathbf{II} is formally a family, but all the members of the family are the same. It is trivial to check that, for all instance u of the problem, $\Pi(s(u)) + \text{cod}(u)$ provides the right solution in one computation step, i.e., it suffices to provide *cod*(u) as input to the unique member of the family in order to obtain the right answer. *Stricto sensu*, it is a solution in the framework of Membrane Computing, but it does not use Membrane Computing strategies. All the work is done in the algorithm A and one can wonder if the computation itself can be performed by using *pure* Membrane Computing techniques.

We focus now on the well-known ST-CONNECTIVITY problem (known as STCON). It can be settled as follows: *Given a directed graph $\langle V, E \rangle$ and two vertices s and t in V , the STCON problem consists of deciding if t is reachable from s , i.e., if there exists a sequence of adjacent vertices (i.e., a path) starting with s and ending with t .* It is known that it is an \mathbf{NL} -complete problem, i.e., it

can be solved by a nondeterministic Turing machine using a logarithmic amount of memory space and every problem in the class **NL** is reducible to STCON under a log-space reduction.

In this paper, we study the STCON in the framework of P systems. As shown above, since $\text{STCON} \in \mathbf{NL} \subseteq \mathbf{P}$, there exist a trivial family of P systems in $\mathbf{PMC}_{\mathcal{F}}$ which solves it, regardless of the model \mathcal{F} . It suffices that \mathcal{F} deals with *send-out* rules. In this paper, we present three designs of uniform families of P systems that solve the decision problem STCON by *pure* Membrane Computing techniques, i.e., techniques where the features of the model \mathcal{F} are exploited in the computation: P systems with membrane creation, P systems with active membranes with dissolution and without polarizations and P systems with active membranes without dissolution and with polarizations. We provide such designs and show the differences with previous studies found in the literature.

Since STCON is **NL**-complete, such designs are *constructive* proofs of the belonging of **NL** to $\mathbf{PMC}_{\mathcal{MC}}$, $\mathbf{PMC}_{\mathcal{AM}_{+d}^0}$ and $\mathbf{PMC}_{\mathcal{AM}_{+d}^+}$.

The paper is structured as follows: First of all, we recall some basic definitions used along the paper. In Section 3, previous works on **NL** are revisited. Next, our designs of solutions are provided and the paper finishes with some conclusions and presenting research lines for a future work.

2 Preliminaries

Next, some basic concepts used along the paper are recalled. We assume that the reader is familiar with Membrane Computing techniques (for a detailed description, see [13]).

A decision problem X is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total Boolean function over I_X . A *P system with input* is a tuple (Π, Σ, i_Π) , where Π is a P system, with working alphabet Γ , with p membranes labelled by $1, \dots, p$, and initial multisets $\mathcal{M}_1, \dots, \mathcal{M}_p$ associated with them; Σ is an (input) alphabet strictly contained in Γ ; the initial multisets are over $\Gamma - \Sigma$; and i_Π is the label of a distinguished (input) membrane. Let (Π, Σ, i_Π) be a P system with input, Γ be the working alphabet of Π , μ its membrane structure, and $\mathcal{M}_1, \dots, \mathcal{M}_p$ the initial multisets of Π . Let m be a multiset over Σ . The *initial configuration of (Π, Σ, i_Π) with input m* is $(\mu, \mathcal{M}_1, \dots, \mathcal{M}_{i_\Pi} \cup m, \dots, \mathcal{M}_p)$. We denote by I_Π the set of all inputs of the P system Π (i.e. I_Π is a collection of multisets over Σ). In the case of P systems with input and *with external output*, the above concepts are introduced in a similar way.

Definition 1. A recognizer P system is a P system with input and with external output such that:

1. The working alphabet contains two distinguished elements *yes*, *no*.
2. All its computations halt.

3. If C is a computation of Π , then either the object *yes* or the object *no* (but not both) must have been released into the environment, and only in the last step of the computation. We say that C is an *accepting computation* (respectively, *rejecting computation*) if the object *yes* (respectively, *no*) appears in the external environment associated to the corresponding halting configuration of C .

3 Previous Works

The relation between the complexity class **NL** and Membrane Computing models has already been explored in the literature. In [6], Murphy and Woods claim that $\mathbf{NL} \subseteq \mathbf{PMC}_{\mathcal{AM}_{d,-u}^0}$, i.e., every problem in the complexity class **NL** can be solved by a non-uniform family of recognizer P systems with active membranes without polarization and without dissolution.

The proof shows the design of a family of P systems with active membranes without polarization and without dissolution which solves STCON and considers the **NL**-completeness of STCON. Nonetheless, the authors use a non standard definition of recognizer P systems. According to the usual definition of *recognizer P system* (see, e.g., [4]), *either one object yes or one object no (but no both) must have been released into the environment, and only in the last step of the computation*. In the proposed family by Murphy and Woods, it is easy to find a P system which sends *yes* to the environment in an intermediate step of the computation and sends *no* to the environment in the last step of the computation, so their proof of $\mathbf{NL} \subseteq \mathbf{PMC}_{\mathcal{AM}_{d,-u}^0}$ cannot be considered valid with respect to the standard definition of recognizer P systems.

Counterexample: Let us consider the instance (s, t, G) of STCON where G has only two vertices s and t and only one edge (s, t) . According to [6], the P system of the cited model that solves this instance has $\Gamma = \{s, t, \text{yes}, \text{no}, c_0, \dots, c_4\}$ as alphabet, h as unique label and $[\]_h$ as membrane structure. The initial configuration is $[sc_4]_h$ and the set of rules consists of the following seven rules:

$$\begin{array}{ll} [s \rightarrow t]_h & [t]_h \rightarrow [\]_h \text{yes} \\ [c_0]_h \rightarrow [\]_h \text{no} & [c_i \rightarrow c_{i-1}]_h \text{ for } i \in \{1, \dots, 4\}. \end{array}$$

It is easy to check that this P system sends *yes* to the environment in the second step of computation and sends *no* in the fifth (and last) step, so, according to the standard definition, it is not a recognizer P system. In [7] Murphy and Woods revisited the solution of STCON by non-uniform families of recognizer P systems and considered three different ways of the acceptance in recognizer P systems, one of them was the standard one (Def. 1).

4 Three Designs for the STCON Problem

In this section, we provide three uniform families of P systems that solve the STCON problem in three different P system models. All these models use the

same encoding of an instance of the problem. We do not loss generality if we consider the n vertices of the graph as $\{1, \dots, n\}$. In this case, a concrete instance $I = (s, t, \langle V, E \rangle)$ of the STCON on a graph $\langle V, E \rangle$ with vertices $\{1, \dots, n\}$, can be encoded as

$$\text{cod}(I) = \{x_s, y_t\} \cup \{a_{ij} : (i, j) \in E\},$$

i.e., x_s stands for the starting vertex, y_t for the ending vertex and a_{ij} for each edge (i, j) in the graph. By using this coding, *all* the instances of the STCON problem with n vertices, can be encoded with the alphabet

$$\begin{aligned} \Sigma = & \{x_i : i \in \{1, \dots, n\}\} \cup \\ & \{y_j : j \in \{1, \dots, n\}\} \cup \\ & \{a_{ij} : i, j \in \{1, \dots, n\}\} \end{aligned}$$

whose cardinality is $2n + n^2$.

Next we present three solutions of the STCON problem by P systems. The first two of them are based on P systems with active membranes, while the last one uses P systems with membrane creation. The first solution does not use membrane dissolution but uses the polarizations of the membranes. The second solution does not use polarizations but uses membrane dissolution instead. Moreover, none of these solutions use membrane division rules.

All the three solutions, roughly speaking, work in the following way. For a given directed graph $G = \langle V, E \rangle$ and vertices s and t , the system creates/activates certain membranes in the initial configuration corresponding to the edges in E . Then, these membranes will be used to create those objects that represent the vertices reachable from s . Meanwhile, it is tested whether or not the vertex t is created or not. If yes, the system initiates a process which will send *yes* out to the environment. If the vertex t is not produced by the system, i.e., t is not reachable from s in G , then a counter will create the symbol *no* which is then sent out to the environment.

Although these solutions are similar, they use different techniques according to the class of P systems that we employ. We believe that some of the constructions used in the following designs might be useful also in solutions of other problems by these classes of P systems.

4.1 P Systems with Active Membranes with Polarization and without Dissolution

As a first approach, we will provide the design of a uniform family $\mathbf{\Pi} = \{II_n\}_{n \in \mathbb{N}}$ of P systems in $\mathbf{PMC}_{\mathcal{AM}-d}$ which solves STCON. Each P system II_n of the family decides on *all the possible* instances of the STCON problem on a graph with n nodes. Such P systems use two polarizations, but they do not use division or dissolution rules, so *not all the types of rules* of P systems with active membranes are necessary to solve STCON. Each II_n will receive as input an instance of the STCON as described above and will release *yes* or *no* into the

environment in the last step of the computation as the answer of the decision problem. The family presented here is

$$II_n = \langle \Gamma_n, \Sigma_n, H_n, EC_n, \mu_n, w_n^a, w_n^1, \dots, w_n^n, w_n^{11}, \dots, w_n^{nn}, w_n^{skin}, \mathcal{R}_n, i_n \rangle.$$

For the sake of simplicity, thereafter we will omit the subindex n . The components of II_n are as follows.

– **Alphabet:**

$$\begin{aligned} \Gamma = & \{x_i, y_i, t_i : i \in \{1, \dots, n\}\} \cup \\ & \{a_{ij}, z_{ij} : i, j \in \{1, \dots, n\}\} \cup \\ & \{c_i : i \in \{0, \dots, 3n + 1\}\} \cup \\ & \{k, yes, no\}. \end{aligned}$$

- **Input alphabet:** Σ , as described at the beginning of the section. Let us remark that $\Sigma \subset \Gamma$.
- **Set of labels:** $H = \{\langle i, j \rangle : i, j \in \{1, \dots, n\}\} \cup \{1, \dots, n\} \cup \{a, skin\}$.
- **Electrical charges:** $EC = \{0, +\}$.
- **Membrane structure:** $[[[]_1^0 \dots []_n^0 []_{\langle 1, 1 \rangle}^0 \dots []_{\langle n, n \rangle}^0 []_a^0]_{skin}^0$.
- **Initial multisets:** $w^a = c_0, w^{skin} = w^{ij} = w^k = \lambda$ for $i, j, k \in \{1, \dots, n\}$.
- **Input label:** $i = skin$.

The set of rules \mathcal{R} :

$$\mathbf{R1.} \quad a_{ij} []_{\langle i, j \rangle}^0 \rightarrow [a_{ij}]_{\langle i, j \rangle}^+ \text{ for } i, j \in \{1, \dots, n\}.$$

Each input object a_{ij} activates the corresponding membrane by changing its polarization. Notice that such a symbol a_{ij} represents an edge in the input graph.

$$\mathbf{R2.} \quad y_j []_j^0 \rightarrow [y_j]_j^+ \text{ for } j \in \{1, \dots, n\}.$$

The object y_j activates the membrane j by changing its polarization. As the input multiset always has exactly one object of the form y_j , II_n will have a unique membrane with label in $\{1, \dots, n\}$ and polarization $+$.

$$\mathbf{R3.} \quad [x_i \rightarrow z_{i1} \dots z_{in} t_i]_{skin}^0 \text{ for } i \in \{1, \dots, n\}.$$

The goal of these rules is to create $n + 1$ copies of an object x_i . A copy z_{ij} will be able to produce an object x_j if the edge (i, j) belongs to E . The object t_i will be used to witness that vertex i is reachable.

$$\mathbf{R4.} \quad \left. \begin{array}{l} z_{ij} []_{\langle i, j \rangle}^+ \rightarrow [x_j]_{\langle i, j \rangle}^0 \\ t_j []_j^+ \rightarrow [k]_j^0 \end{array} \right\} \text{ for } i, j \in \{1, \dots, n\}.$$

If the membrane with label $\langle i, j \rangle$ has polarization $+$, then the symbol z_{ij} produces a symbol x_j inside this membrane. Meanwhile, the polarization of this membrane changes from $+$ to 0 , i.e., the membrane is deactivated. Moreover, if the symbol t_j appears in the skin and the membrane with label j has positive polarization, then an object k is produced inside this membrane. Such object k will start the process to send out *yes* to the environment.

$$\mathbf{R5.}, \quad [k]_j^0 \rightarrow k []_j^0 \quad k []_a^0 \rightarrow [k]_a^+.$$

The object k is a witness of the success of the STCON problem. If it is produced, it goes into the membrane with label a and changes its polarization to $+$.

R6. $[x_j]_{\langle i,j \rangle}^0 \rightarrow x_j []_{\langle i,j \rangle}^0$ for $i, j \in \{1, \dots, n\}$.

The produced object x_j is sent to the membrane *skin* in order to continue the computation by rules form **R3**.

R7. $\left. \begin{array}{l} [c_i \rightarrow c_{i+1}]_a^0, [c_{3n+1}]_a^0 \rightarrow no []_a^0 \\ [c_i \rightarrow c_{i+1}]_a^+, [c_{3n+1}]_a^+ \rightarrow yes []_a^0 \end{array} \right\}$ for $i \in \{0, \dots, 3n\}$.

Object c_i evolves to c_{i+1} regardless of the polarization of the membrane with label a . If during the evolution the object k enters the membrane with label a , then the polarization of this membrane changes to $+$ and the object c_{3n+1} will produce *yes* in the skin membrane. Otherwise, if the object k is not produced, the polarization is not changed and the object c_{3n+1} will produce *no*.

R8. $[no]_{skin} \rightarrow no []_{skin}, [yes]_{skin} \rightarrow yes []_{skin}$.

Finally, *yes* or *no* is sent out the P system in the last step of computation.

To see in more details how a computation of the presented P system goes, let us consider an instance $I = (s, t, G)$ of STCON where G is a graph $\langle \{1, \dots, n\}, E \rangle$. The computation of Π_n on $cod(I)$ can be described as follows. During the first step, using rules in **R1**, every a_{ij} enters the membrane with label $\langle i, j \rangle$ and changes its polarization to $+$. Thus, after the first step the edges in E are encoded by the positive polarizations of the membranes with labels of the form $\langle i, j \rangle$. During the same step, using the corresponding rule in **R2**, y_t enters the membrane with label t and changes its polarization to $+$. This membrane will be used to recognize if an object representing that t is reachable from s is introduced by the system.

Now let $l \in \{1, 4, \dots, 3(n-1) + 1\}$ and consider an object x_i in the skin membrane. During the l th step, using rules in **R3**, x_i creates $n + 1$ copies of itself. The system will try to use a copy z_{ij} ($j \in \{1, \dots, n\}$) in the next step to create a new object x_j . The copy t_i will be used to decide if $i = t$.

During the $(l + 1)$ th step, using rules in **R4**, the systems sends z_{ij} into the membrane with label $\langle i, j \rangle$ if that membrane has a positive polarization. Meanwhile, z_{ij} evolves to x_j and the polarization of the membrane changes to neutral. During the same step, if $i = t$ and the membrane with label t has positive polarization, then the system sends t_i into this membrane. Meanwhile, t_i evolves to k and the polarization of the membrane changes to neutral.

During the $(l + 2)$ th step, using rules in **R6**, the object x_j is sent out of the membrane with label $\langle i, j \rangle$. Moreover, if the membrane with label t contains k , then this k is sent out of this membrane.

One can see that during the above three steps the system introduces an object x_j if and only if (i, j) is an edge in E . Using this observation we can derive that during the computation of the system, an object x_j appears in the skin if and only if there is a path in G from s to j . Thus, t is reachable from s in G if and only if there is a configuration of Π_n where the skin contains x_t . However, in this case an object k is introduced in the membrane with label t . It can also be seen that Π_n sends out to the environment *yes* if and only if k appears in membrane t . Moreover, if k does not appear in membrane t , then the systems sends out to the environment *no*. Thus, Π_n sends out to the environment *yes* or *no* according to that t is reachable from s or not. As Π_n stops in at most $3n + 2$

steps, we can conclude that the family $\mathbf{\Pi}$ decides STCON in linear time in the number of vertices of the input graph.

4.2 P Systems with Active Membranes with Dissolution and without Polarization

Based on the solution presented in the previous sub-section, we give here a uniform family $\mathbf{\Pi} = \{\Pi_n\}_{n \in \mathbb{N}}$ in $\mathbf{PMC}_{\mathcal{AM}^0}$ which solves STCON. As here we cannot use the polarizations of the membranes, we use membrane dissolution to select those membranes of the initial configuration that correspond to the edges of the input graph. Next we will describe the mentioned family $\mathbf{\Pi}$. Since we do not use polarizations, we do not indicate it at the upper-right corner of the membranes. The family presented here is

$$\Pi_n = \langle \Gamma, \Sigma, H, EC, \mu, W, \mathcal{R}, i \rangle.$$

The components of Π_n are as follows.

– **Alphabet:**

$$\begin{aligned} \Gamma = & \{x_i, v_{1i}, v_{2i}, v_{3i}, v_i, y_i, t_i : i \in \{1, \dots, n\}\} \cup \\ & \{a_{ij}, z_{ij} : i, j \in \{1, \dots, n\}\} \cup \\ & \{c_i : i \in \{0, \dots, 3n + 4\}\} \cup \\ & \{k, yes, no\}. \end{aligned}$$

– **Input alphabet:** Σ , as described at the beginning of the section.

– **Set of labels:** $H = \{\langle i, j, in \rangle, \langle i, j, out \rangle : i, j \in \{1, \dots, n\}\} \cup \{\langle i, in \rangle, \langle i, out \rangle : i \in \{1, \dots, n\}\} \cup \{a, skin\}$.

– **Electrical charges:** $EC = \emptyset$.

– **Membrane structure:** $[[[\langle 1, in \rangle]_{\langle 1, out \rangle} \cdots [[[\langle n, in \rangle]_{\langle n, out \rangle} [[[\langle 1, 1, in \rangle]_{\langle 1, 1, out \rangle} \cdots \cdots [[[\langle n, n, in \rangle]_{\langle n, n, out \rangle} [a]_{skin}$.

– **Initial multisets:** $W = \{w^a, w^{\langle 1, in \rangle}, \dots, w^{\langle n, in \rangle}, w^{\langle 1, out \rangle}, \dots, w^{\langle n, out \rangle}, w^{\langle 1, 1, in \rangle}, \dots, w^{\langle n, n, in \rangle}, w^{\langle 1, 1, out \rangle}, \dots, w^{\langle n, n, out \rangle}, w^{skin}\}$, where $w^a = c_0$, $w^{skin} = w^{\langle i, j, out \rangle} = w^{\langle k, out \rangle} = \lambda$, $w^{\langle i, j, in \rangle} = w^{\langle k, in \rangle} = f_0$, for $i, j, k \in \{1, \dots, n\}$.

– **Input label:** $i = skin$.

The set of rules \mathcal{R} :

R0. $[x_i \rightarrow v_{1i}]_{skin}, [v_{ji} \rightarrow v_{j+1, i}]_{skin}, [v_{3i} \rightarrow v_i]_{skin}$ for $i \in \{1, \dots, n\}$ and $j \in \{1, 2\}$.

In this solution we cannot use an object x_i in the same role as we did in the previous sub-section because of the following reason. The system needs four steps to select those membranes in the initial membrane configuration that correspond to the edges in E . Thus, the system introduces in four steps the object v_i which will act in this solution as x_i did in the previous one.

R1. $\left. \begin{array}{l} [f_m \rightarrow f_{m+1}]_{\langle i, j, in \rangle} \\ [f_3]_{\langle i, j, in \rangle} \rightarrow f_4 \\ [f_4]_{\langle i, j, out \rangle} \rightarrow f_4 \end{array} \right\}$ for $i, j \in \{1, \dots, n\}, m \in \{0, 1, 2\}$.

These rules can dissolve the membranes with label $\langle i, j, in \rangle$ or $\langle i, j, out \rangle$. However, if a_{ij} is in the input multiset, then it prevents the dissolution of the membrane with label $\langle i, j, out \rangle$ using the following rules.

$$\mathbf{R2.} \quad \left. \begin{array}{l} a_{ij} []_{\langle i, j, m \rangle} \rightarrow [a_{ij}]_{\langle i, j, m \rangle} \\ [a_{ij}]_{\langle i, j, in \rangle} \rightarrow a_{ij} \end{array} \right\} \text{ for } i, j \in \{1, \dots, n\}, m \in \{in, out\}.$$

By these rules the input symbol a_{ij} goes into the membrane with label $\langle i, j, in \rangle$ and dissolves that. This way the second rule in **R1** cannot be applied, thus the membrane with label $\langle i, j, out \rangle$ cannot be dissolved by the third rule.

$$\mathbf{R3.} \quad \left. \begin{array}{l} [f_m \rightarrow f_{m+1}]_{\langle j, in \rangle} \\ [f_3]_{\langle j, in \rangle} \rightarrow f_4 \\ [f_4]_{\langle j, out \rangle} \rightarrow f_4 \end{array} \right\} \text{ for } j \in \{1, \dots, n\}, m \in \{0, 1, 2\}.$$

These rules can dissolve the membranes with label $\langle j, in \rangle$ or $\langle j, out \rangle$. However, if y_j is in the input multiset, then it prevents the dissolution of the membrane with label $\langle j, out \rangle$ using the following rules.

$$\mathbf{R4.} \quad \left. \begin{array}{l} y_j []_{\langle j, m \rangle} \rightarrow [y_j]_{\langle j, m \rangle} \\ [y_j]_{\langle j, in \rangle} \rightarrow y_j \end{array} \right\} \text{ for } j \in \{1, \dots, n\} \text{ and } m \in \{in, out\}.$$

By these rules the input symbol y_j goes into the membrane with label $\langle j, in \rangle$ and dissolves that. With this it is achieved that the membrane with label $\langle j, out \rangle$ is not dissolved by the rules in **R3**.

$$\mathbf{R5.} \quad [v_i \rightarrow z_{i1} \dots z_{in} t_i]_{skin} \text{ for } i \in \{1, \dots, n\}.$$

The role of these rules is the same as that of the rules in **R3** in Section 4.1.

$$\mathbf{R6.} \quad \left. \begin{array}{l} z_{ij} []_{\langle i, j, out \rangle} \rightarrow [v_j]_{\langle i, j, out \rangle} \\ t_j []_{\langle j, out \rangle} \rightarrow [k]_{\langle j, out \rangle} \end{array} \right\} \text{ for } i, j \in \{1, \dots, n\}.$$

The role of these rules is similar to that of the rules in **R4** in Section 4.1: If the membrane with label $\langle i, j, out \rangle$ has not been dissolved, then the object z_{ij} produces a symbol v_j inside this membrane. Analogously, if the symbol t_j appears in the skin and the membrane with label $\langle j, out \rangle$ is not dissolved, then an object k is produced inside this membrane. Such object k will start the process to send *yes* out to the environment.

$$\mathbf{R7.} \quad [k]_{\langle j, out \rangle} \rightarrow k []_{\langle j, out \rangle}, \quad k []_a \rightarrow [k]_a, \quad [k]_a \rightarrow k.$$

The object k is a witness of the success of the STCON problem. If it is produced, it goes into the membrane with label a and dissolves it.

$$\mathbf{R8.} \quad [v_j]_{\langle i, j \rangle} \rightarrow v_j \text{ for } i, j \in \{1, \dots, n\}.$$

The produced object v_j dissolves the membrane with label $\langle i, j \rangle$ as the computation does not need this membrane any more. This way the object v_j appears in the skin and the computation can continue using the rules in **R5**.

$$\mathbf{R9.} \quad \left. \begin{array}{l} [c_i \rightarrow c_{i+1}]_a, [c_{3n+4}]_a \rightarrow no []_a \\ [c_{i+1}]_{skin} \rightarrow [yes]_{skin} \end{array} \right\} \text{ for } i \in \{0, \dots, 3n+3\}.$$

Object c_i evolves to c_{i+1} in membrane with label a . If during this evolution the object k appears in this membrane, then it dissolves it and the object c_{i+1} gets into the skin membrane where it produces *yes*. Otherwise, if the object k is not produced, c_{3n+4} remains in membrane with label a and produces *no*.

$$\mathbf{R10.} \quad [no]_{skin} \rightarrow no []_{skin}, \quad [yes]_{skin} \rightarrow yes []_{skin}.$$

Finally, *yes* or *no* is sent out the P system in the last step of computation.

One can observe that during the first four steps of Π_n a membrane with label $\langle i, j, out \rangle$ is not dissolved if and only if a_{ij} is in the input. Thus, Π_n has a membrane with label $\langle i, j, out \rangle$ after the first four steps if and only if Π_n defined in Section 4.1 has a membrane $\langle i, j \rangle$ with positive polarization after the first step. Similar observations apply in the case of membranes with label $\langle j, out \rangle$. Thus, the correctness of Π_n defined in this section follows from the correctness of Π_n defined in Section 4.1. One can also observe that Π_n stops after at most $3n + 5$ steps, which means that the family Π defined in this section decides STCON in linear time.

4.3 P Systems with Membrane Creation

Here we provide a solution of the problem STCON by a uniform family of P systems in the framework of *P systems with Membrane Creation*. Since STCON is **NL**-complete, we have a direct proof of $\mathbf{NL} \subseteq \mathbf{PMC}_{\mathcal{MC}}$. This result is well-known, since $\mathbf{NL} \subset \mathbf{NP}$ and $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{MC}}$ (see [4]). Nonetheless, to the best of our knowledge, this is the first design of a P system family which solves STCON in $\mathbf{PMC}_{\mathcal{MC}}$.

Next we will describe the family $\Pi = \{\Pi_n\}_{n \in \mathbb{N}}$ of P systems in $\mathbf{PMC}_{\mathcal{MC}}$. Each Π_n will receive as input an instance of the STCON as described at the beginning of the section and will release *yes* or *no* into the environment in the last step of the computation as the answer of the decision problem. The family presented here is

$$\Pi_n = \langle \Gamma, \Sigma, H, \mu, w^a, w^b, w^c, \mathcal{R}, i \rangle.$$

The components of Π_n are as follows.

– **Alphabet:**

$$\begin{aligned} \Gamma = & \{x_i, y_i, t_i : i \in \{1, \dots, n\}\} \cup \\ & \{a_{ij}, z_{ij} : i, j \in \{1, \dots, n\}\} \cup \\ & \{no_i : i \in \{0, \dots, 3n + 3\}\} \cup \\ & \{yes_i : i \in \{1, \dots, 4\}\} \cup \\ & \{yes, no\}. \end{aligned}$$

- **Input alphabet:** Σ , as it is described at beginning of the section.
- **Set of labels:** $H = \{\langle i, j \rangle : i, j \in \{1, \dots, n\}\} \cup \{1, \dots, n\} \cup \{a, b, c\}$.
- **Membrane structure:** $[[[]_a []_b]_c]$.
- **Initial multisets:** $w^a = no_0, w^b = w^c = \lambda$.
- **Input label:** $i = b$.

The set of rules \mathcal{R} :

R1. $[[a_{ij} \rightarrow [\lambda]_{\langle i, j \rangle}]_b]$ for $i, j \in \{1, \dots, n\}$.

Each input symbol a_{ij} creates a new membrane with label $\langle i, j \rangle$. Recall that such a symbol a_{ij} represents an edge in the directed graph.

R2. $[y_j \rightarrow [\lambda]_j]_b]$ for $j \in \{1, \dots, n\}$.

By these rules an input symbol y_j creates a new membrane with label j .

R3. $[x_i \rightarrow z_{i_1} \dots z_{i_n} t_i]_b]$ for $i \in \{1, \dots, n\}$.

The role of these rules is the same as that of the rules in **R3** in Section 4.1.

$$\mathbf{R4.} \quad \left. \begin{array}{l} z_{ij} []_{\langle i,j \rangle} \rightarrow [x_j]_{\langle i,j \rangle} \\ t_j []_j \rightarrow [yes_0]_j \end{array} \right\} \text{ for } i, j \in \{1, \dots, n\}.$$

The role of these rules is similar to that of the rules in **R4** in Section 4.1 except that here an object t_j introduces an object yes_0 in the membrane with label j . This new object yes_0 will evolve with the rules in **R6** and **R7** until the final object yes is produced in the environment.

$$\mathbf{R5.} \quad [x_j]_{\langle i,j \rangle} \rightarrow x_j \text{ for } i, j \in \{1, \dots, n\}.$$

The object x_j dissolves the membrane with label $\langle i, j \rangle$. The useful information is that x_j is reachable. We keep this information, but the membrane can be dissolved. This way x_j gets to the membrane b and the computation can go on using the rules in **R3**.

$$\mathbf{R6.} \quad [yes_0]_j \rightarrow yes_1 \text{ for } j \in \{1, \dots, n\}.$$

For each possible value of j , if yes_0 is produced, the corresponding membrane is dissolved and yes_1 appears in the membrane with label b .

$$\mathbf{R7.} \quad \begin{array}{l} [yes_1]_b \rightarrow yes_2, yes_2 []_a \rightarrow [yes_3]_a, \\ [yes_3]_a \rightarrow yes_4, [yes_4]_c \rightarrow yes []_c. \end{array}$$

The evolution of the object yes_i firstly dissolves the membrane with label b . If this membrane is dissolved, the rules from **R3** will be no longer applied. In a similar way, object yes_3 dissolves the membrane with label a and this stops the evolution of the objects inside this membrane.

$$\mathbf{R8.} \quad [no_i \rightarrow no_{i+1}]_a \text{ for } i \in \{1, \dots, 3n + 2\}.$$

The object no_i evolves inside the membrane with label a . If this evolution is not halted by the dissolution of this membrane, these objects will produce an object no in the environment.

$$\mathbf{R9.} \quad [no_{3n+3}]_a \rightarrow no, [no]_c \rightarrow no []_c.$$

If the evolution of no_i is not stopped, the object no_{3n+3} dissolves the membrane with label a and creates a new object no . This object will be sent to the environment in the next step of the computation.

It is not difficult to see using the comments given after the rules that this solution works essentially in the same way as our first solution. The main difference is that while in Section 4.1 an input symbol a_{ij} is used to change the polarization of a membrane $\langle i, j \rangle$, here this symbol is used to create such a membrane. Thus, the correctness of the solution presented here can be seen using the correctness of the solution given in Section 4.1. It is also clear that the P systems presented here work in linear time in the number of vertices in the input graph.

As we have mentioned, in solutions of problems in **P** via uniform families of P systems it is important to use such input encoding and P system constructing devices that are not capable to compute the correct answer. It is easy to see that the decision processes in the solutions of STCON presented in this paper are entirely done by the P systems themselves. Thus our solutions could be easily modified so that the construction of the used families and the computation of the input encoding can be carried out by reasonable weak computational devices, for example, by logarithmic-space deterministic Turing machines.

5 Conclusions

The design of a uniform family of recognizer P systems working in polynomial time which solves a decision problem with *pure* Membrane Computing techniques is a hard task, regardless of the complexity class of the problem. The difficulty comes from the hard restrictions imposed on such family. Firstly, the use of *input* P systems implies that each instance of the problem must be encoded as a multiset and such multiset must be introduced at the starting configuration in *one* input membrane. The multiset encoding the instance cannot be distributed in several membranes in the starting configuration. Secondly, in *uniform* families, each P system must solve *all* the instances of the problem of the same *size* (regardless of whether the answer is positive or not). This means that the set of rules which leads to send *yes* to the environment and the set of rules which leads to send *no* must be present in the design of the P system; and thirdly, the standard definition of recognizer P systems claims that an object *yes* or *no* (but no both) is sent to the environment in the *last* step of computation.

A deep study of these constraints shows that it is not sufficient to implement a design of P system with the control scheme “*if* the restrictions of the decision problem are satisfied, *then* an object *yes* must be sent to the environment”. Instead of such scheme, the design must consider the following structure: “*if* the restrictions are satisfied, *then* an object *yes* must be sent to the environment, *else* an object *no* must be sent”. This scheme *if-then-else* must be controlled with the ingredients of the P system model. In the three presented designs, this *if-then-else* scheme is implemented via dissolution, polarization, or membrane creation.

These ideas lead us to consider the necessity of revisiting the complexity classes under **P** and adapt the definition of recognizer P systems for these classes. Some papers in this new research line can be found in the literature (see, e.g., [12]), but further research is needed.

Acknowledgements

The authors gratefully acknowledge the helpful suggestions and comments of the anonymous referees. This work was partially done during Zsolt Gazdag’s visit at the Research Institute of Mathematics of the University of Sevilla (IMUS) partially supported by IMUS. Miguel A. Gutiérrez–Naranjo acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

1. Csuhaĵ-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.): Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers, Lecture Notes in Computer Science, vol. 7762. Springer (2013)

2. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A Linear Time Solution to the Partition Problem in a Cellular Tissue-Like Model. *Journal of Computational and Theoretical Nanoscience* 7(5), 884–889 (MAY 2010)
3. Gazdag, Z., Kolonits, G.: A new approach for solving SAT by P systems with active membranes. In: Csuhaj-Varjú et al. [1], pp. 195–207
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. *Theoretical Computer Science* 371(1-2), 54–61 (2007)
5. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Workshop on Membrane Computing. Lecture Notes in Computer Science*, vol. 4860, pp. 336–352. Springer, Berlin Heidelberg (2007)
6. Murphy, N., Woods, D.: A characterisation of NL using membrane systems without charges and dissolution. In: Calude, C.S., Costa, J.F., Freund, R., Oswald, M., Rozenberg, G. (eds.) *Unconventional Computing, Lecture Notes in Computer Science*, vol. 5204, pp. 164–176. Springer Berlin Heidelberg (2008)
7. Murphy, N., Woods, D.: On acceptance conditions for membrane systems: characterisations of L and NL. In: *Proceedings International Workshop on The Complexity of Simple Programs*. Cork, Ireland, 6-7th December 2008. pp. 172–184. *Electronic Proceedings in Theoretical Computer Science*. Vol 1 (2009)
8. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun et al. [13], pp. 302 – 336
9. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. In: Csuhaj-Varjú, E., Kintala, C., Wotschke, D., Vaszil, G. (eds.) *Proceeding of the 5th Workshop on Descriptive Complexity of Formal Systems. DCFS 2003*. pp. 284–294 (2003)
10. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics* 11(4), 423–434 (2006)
11. Porreca, A.E.: *Computational Complexity Classes for Membrane System*. Master's thesis, Universtita' di Milano-Bicocca, Italy (2008)
12. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú et al. [1], pp. 342–357
13. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England (2010)

Simulating Turing Machines with Polarizationless P Systems with Active Membranes

Zsolt Gazdag¹, Gábor Kolonits¹, Miguel A. Gutiérrez–Naranjo²

¹ Department of Algorithms and their Applications
Faculty of Informatics

Eötvös Loránd University, Budapest, Hungary

² Research Group on Natural Computing

Department of Computer Science and Artificial Intelligence

University of Sevilla, 41012, Spain

{gazdagzs, kolomax}@inf.elte.hu, magutier@us.es

Abstract. In this paper, we prove that every single-tape deterministic Turing machine working in time $t(n)$, for some function $t : \mathbb{N} \rightarrow \mathbb{N}$, can be simulated by a uniform family of polarizationless P systems with active membranes. Moreover, this is done without significant slowdown in the working time. Furthermore, if $\log t(n)$ is space constructible, then the members of the uniform family can be constructed by a family machine that uses space $O(\log t(n))$.

1 Introduction

The simulation of the behaviour of Turing machines by families of P systems has a long tradition in Membrane Computing (see, e.g., [1, 11, 8, 13]). The purpose of such simulations is twofold. On the one hand, they allow to prove new properties on complexity classes and, on the other hand, they provide constructive proofs of results which have been proved via indirect methods³.

In this paper, we give a new step on the second research line, by showing that Turing machines can be simulated efficiently by families of polarizationless P systems with active membranes. By efficiency we mean that these P systems can simulate Turing machines without significant slowdown in the working time. Moreover, the space complexity of the presented P systems is quadratic in the time complexity of the Turing machine.

The conclusions obtained from such simulations are well-known: the decision problems solved by Turing machines can also be solved by families of devices in the corresponding P system models. In fact, it is well-known that the solution of a decision problem X belonging to the complexity class \mathbf{P} via a polynomially uniform family of recognizer P systems is trivial, since the polynomial encoding of the input can involve the solution of the problem (see [3, 7]).

³ The reader is supposed to be familiar with standard techniques and notations used in Membrane Computing. For a detailed description see [10].

This fact can be generalized to more wide situations: the solution of a decision problem X by a uniform family of P systems Π may be trivial in the following sense. Let us consider a Turing machine that computes the encoding of the instances of X (also called the *encoding machine*). If this machine is powerful enough to decide if an instance is a positive instance of X or not, then a trivial P system can be used to send out to the environment the correct answer.

In order to avoid such trivial solutions, the encoding machine and the Turing machine that computes the members of Π (often called the *family machine*) should be reasonably weak. More precisely, if the problem X belongs to a complexity class \mathcal{C} , then the family machine and the encoding machine should belong to a class of Turing machines that can compute only a strict subclass of \mathcal{C} (see [8]).

According to this, we will use a family of P systems to simulate a Turing machine M which members can be constructed by a family machine with the following property. If M works in time $t(n)$, for some function $t : \mathbb{N} \rightarrow \mathbb{N}$, and $\log t(n)$ is space constructable, then the family machine works using space $O(\log t(n))$. In particular, if t is a polynomial, then the family machine uses logarithmic space. Moreover, we will use the following function pos to encode the input words of M : For a given input word w , $pos(w)$ is a multiset where every letter of w is coupled with its position in w . Furthermore, the positions of the letters are encoded in binary words. It was discussed in [8] that pos is computable by deterministic random-access Turing machines using logarithmic time (in other words, pos is **DLOGTIME** computable). In this way, there is no risk that pos can compute a solution of a problem outside of **DLOGTIME**. Since **DLOGTIME** is a rather small complexity class, it follows that we can use pos safely as the input encoding function during the simulation of M .

The presented result is similar to the one appearing in [1] stating that every single-tape deterministic Turing machine can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. However, this result and ours are not comparable, as the constructions in [1] use the polarizations of the membranes, while our solution does not.

The paper is organized as follows. First of all, we recall some basic definitions used along the paper. Then, in Section 3, we present the main result. Finally, we give some concluding remarks in Section 4.

2 Preliminaries

First, we recall some basic concepts used later.

Alphabets, Words, Multisets. An *alphabet* Σ is a non-empty and finite set of symbols. The elements of Σ are called *letters* and Σ^* denotes the set of all finite *words* (or *strings*) over Σ , including the *empty word* ε . The length of a word $w \in \Sigma^*$ is denoted by $l(w)$. We will use *multisets* of objects in the membranes of a P system. As usual, these multisets will be represented by strings over the object alphabet of the P system.

The set of natural numbers is denoted by \mathbb{N} . For $i, j \in \mathbb{N}$, $[i, j]$ denotes the set $\{i, i+1, \dots, j\}$ (notice that if $j < i$, then $[i, j] = \emptyset$). For the sake of simplicity, we will write $[n]$ instead of $[1, n]$. For a number $i \in \mathbb{N}$, $b(i)$ denotes its binary form and $b(\mathbb{N}) = \{b(i) \mid i \in \mathbb{N}\}$. Given an alphabet Σ , the function $pos : \Sigma^* \rightarrow (\Sigma \times b(\mathbb{N}))^*$ is defined in the following way. For a word $w = a_1 \dots a_n \in \Sigma^*$, where $a_i \in \Sigma, i \in [n]$, $pos(w) := (a_1, b(1)) \dots (a_n, b(n))$. If a is the i th letter of w , then we will also write the i th letter of $pos(w)$ in the form $a_{b(i)}$.

Turing Machines. Turing machines are well known computational devices. In the following we describe the variant that appears, e.g., in [12]. A (*deterministic*) Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

- Q is the finite set of *states*,
- Σ is the *input alphabet*,
- Γ is the tape alphabet including Σ and a distinguished symbol $\sqcup \notin \Sigma$, called the *blank symbol*,
- $\delta : (Q - \{q_a, q_r\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the *transition function*,
- $q_0 \in Q$ is the *initial state*,
- $q_a \in Q$ is the *accepting state*,
- $q_r \in Q$ is the *rejecting state*.

M works on a single infinite tape that is closed on the left-hand side. During the computation of M , the tape contains only finitely many non-blank symbols, and it is blank everywhere else. Let us consider a word $w \in \Sigma^*$. The initial configuration of M on w is the configuration where w is placed at the beginning of the tape, the head points to the first letter of w , and the current state of M is q_0 . A configuration step performed by M can be described as follows. If M is in state p and the head of M reads the symbol X , then M can change its state to q and write X' onto X if and only if $\delta(p, X) = (q, X', d)$, for some $d \in \{L, R\}$. Moreover, if $d = R$ (resp. $d = L$), then M moves its head one cell to the right (resp. to the left) (as usual, M can never move the head off the left-hand end of the tape even if the head points to the first cell and $d = L$). We say that M accepts (resp. rejects) w , if M can reach from the initial configuration on w the accepting state q_a (resp. the rejecting state q_r). Notice that M can stop only in these states. The language accepted by M is the set $L(M)$ consisting of those words in Σ^* that are accepted by M . It is said that M works in time $t(n)$ ($t : \mathbb{N} \rightarrow \mathbb{N}$) if, for every word $w \in \Sigma^*$, w stops on w after at most $t(l(w))$ steps; M works using space $s(n)$ ($s : \mathbb{N} \rightarrow \mathbb{N}$) if it uses at most $s(n)$ cells when it is started on an input word with length n . As usual, if M is a multi-tape Turing machine and it does not write any symbol on its input tape, then those cells that are used to hold the input word are not counted when the space complexity of M is measured⁴. Let us consider a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $f(n)$ is at least $O(\log n)$. We say that f is *space constructible* if there is a Turing machine M that

⁴ For the formal definitions of the well known complexity classes concerning Turing machines (such as **L**, **P**, **TIME**($t(n)$) and **SPACE**($s(n)$)), the interested reader is referred to [12].

works using space $O(f(n))$ and M always halts with the unary representation of $f(n)$ on its tape when started on input 1^n .

Recognizer P systems. A P system is a construct of the form $\Pi = (\Gamma, H, \mu, w_1, \dots, w_m, R)$, where $m \geq 1$ (the *initial degree* of the system); Γ is the *working alphabet of objects*; H is a finite set of *labels* for membranes; μ is a *membrane structure* (a rooted tree), consisting of m membranes, labelled with elements of H ; w_1, \dots, w_m are strings over Γ , describing the *initial multisets of objects* placed in the m regions of μ ; and R is a finite set of *developmental rules*.

A P system with input is a tuple (Π, Σ, i_0) , where Π is a P system with working alphabet Γ , with m membranes, and initial multisets w_1, \dots, w_m associated with them; Σ is an (input) alphabet strictly contained in Γ ; the initial multisets are over $\Gamma - \Sigma$; and i_0 is the label of a distinguished (input) membrane.

We say that Π is a *recognizer P system* [4, 5] if Π is a P system with input alphabet Σ and working alphabet Γ ; Γ has two designated objects *yes* and *no*; every computation of Π halts and sends out to the environment either *yes* or *no*, but not both, and this is done exactly in the last step of the computation; and, for a word $w \in \Sigma^*$, called the *input of Π* , w can be added to the system by placing it into the input membrane i_0 in the initial configuration.

A P system Π is *deterministic* if it has only a single computation from its initial configuration to its unique halting configuration. Π is *confluent* if every computation of Π halts and sends out to the environment the same object. Notice that, by definition, recognizing P systems are confluent.

P Systems with Active Membranes. In this paper, we investigate recognizer P systems with active membranes [9]. These systems have the following types of rules. As we are dealing with P systems that do not use the polarizations of the membranes, we leave out this feature from the definition.

- (a) $[a \rightarrow v]_h$, for $h \in H, a \in \Gamma, v \in \Gamma^*$
(object evolution rules, associated with membranes and depending on the label of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);
- (b) $a[]_h \rightarrow [b]_h$, for $h \in H, a, b \in \Gamma$
(*send-in* communication rules, sending an object into a membrane, maybe modified during this process);
- (c) $[a]_h \rightarrow []_h b$, for $h \in H, a, b \in \Gamma$
(*send-out* communication rules; an object is sent out of the membrane, maybe modified during this process);
- (d) $[a]_h \rightarrow b$, for $h \in H, a, b \in \Gamma$
(membrane dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);
- (e) $[a]_h \rightarrow [b]_h [c]_h$, for $h \in H, a, b, c \in \Gamma$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes; the object a specified in the rule

is replaced in the two new membranes by (possibly new) objects b and c respectively, and the remaining objects are duplicated; the new membranes have the same labels as the divided one).

As usual, a P system with active membranes works in a *maximally parallel* manner:

- In one step, any object of a membrane that can evolve must evolve, but one object can be used by only one rule in (a)-(e);
- when some rules in (b)-(e) can be applied to a certain membrane, then one of them must be applied, but a membrane can be the subject of only one of these rules during each step.

We will use uniform families of P system to decide a language $L \subseteq \Sigma^*$. In this paper, we follow the notion of uniformity used in [8] for the definition of *uniform* families of P systems: Let E and F be classes of computable functions. A family $\Pi = (\Pi(i))_{i \in \mathbb{N}}$ of recognizing P systems is called (E, F) -*uniform* if and only if (i) there is a function $f \in F$ such that, for every $n \in \mathbb{N}$, $\Pi(n) = f(1^n)$ (i.e., mapping the unary representation of each natural number to an encoding of the P system processing all the inputs of length n); (ii) there is a function $e \in E$ that maps every word $x \in \Sigma^*$ to a multiset $e(x) = w_x$ over the input alphabet of $\Pi(l(x))$.

An (E, F) -*uniform* family of P systems $\Pi = (\Pi(i))_{i \in \mathbb{N}}$ *decides a language* $L \subseteq \Sigma^*$ if, for every word $x \in \Sigma^*$, starting $\Pi(l(x))$ with w_x in its input membrane, $\Pi(l(x))$ sends out to the environment *yes* if and only if $x \in L$. In general, E and F are well known complexity classes such as **P** or **L**.

We say that $\Pi(n)$ *works in time* $t(n)$ ($t : \mathbb{N} \rightarrow \mathbb{N}$) if $\Pi(n)$ halts in at most $t(n)$ steps, for every input multiset in its input membrane. Next, we adopt the notion of space complexity for families of recognizer P systems similarly to the definition appearing in [8] (see also [6]). Let \mathcal{C} be a configuration of a P system Π . The *size of* \mathcal{C} (denoted by $|\mathcal{C}|$) is the sum of the number of membranes and the total number of objects in \mathcal{C} . If $\mathcal{C} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$ is a halting computation of Π , then the space required by \mathcal{C} is defined as $|\mathcal{C}| = \max\{|\mathcal{C}_0|, \dots, |\mathcal{C}_k|\}$. The space required by Π is $|\Pi| = \sup\{|\mathcal{C}| \mid \mathcal{C} \text{ is a halting computation of } \Pi\}$. Let us note that in the presented solution a P system Π always has finitely many different halting computations, which clearly implies that $|\Pi| \in \mathbb{N}$. Finally, $\Pi(n)$ *works using space* $s(n)$ ($s : \mathbb{N} \rightarrow \mathbb{N}$), if $|\Pi(n)| \leq s(n)$, for every input multiset in its input membrane.

3 The Main Results

In this section we prove the main result of the paper:

Theorem 1. *Let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a function such that $\log t(n)$ is space constructible and consider a Turing machine M working in time $t(n)$. Then M can be simulated by a $(\mathbf{DLOGTIME}, \mathbf{SPACE}(\log t(n)))$ -uniform family $\Pi_M = (\Pi_M(i))_{i \in \mathbb{N}}$ of recognizer P systems with the following properties:*

- the members of Π_M are polarizationless P systems with active membranes, without using membrane division rules, and
- for every $n \in \mathbb{N}$, $\Pi_M(n)$ works in time $O(t(n))$ and in space $O(t^2(n))$.

The rest of this section is devoted to the proof of this theorem. Let us consider a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ working in time $t(n)$. We construct a uniform family of recognizer P systems $\Pi_M = (\Pi_M(i))_{i \in \mathbb{N}}$ that decides the language $M(L)$. Assume that $Q = \{s_1, \dots, s_m\}$, for some $m \geq 3$, where $s_1 = q_0$, $s_{m-1} = q_a$ and $s_m = q_r$. Moreover, $\Gamma = \{X_1, \dots, X_k\}$ for some $k > |\Sigma|$, where $X_k = \sqcup$ is the blank symbol of the working tape.

Before giving the precise construction of $\Pi_M(n)$, we describe informally some of its components. As the number of certain components of $\Pi_M(n)$ will depend on n , when the family machine that constructs $\Pi_M(n)$ enumerates these components, an efficient representation of numbers depending on n should be used. Thus, instead of using a number to denote a component of $\Pi_M(n)$, we will use the binary form of this number.

As M stops in at most $t(n)$ steps, the segment of the tape of M that is used during its work consists of at most $t(n)$ cells. This segment of the tape will be represented by the nested membrane structure appearing on Fig. 1. Here the first membrane in the skin represents the first cell of the tape, while the innermost membrane represents the $t(n)$ th one. We will call these membranes of $\Pi_M(n)$ *tape-membranes*. Let us consider a tape-membrane representing the l th cell of the tape. We call this membrane the l th *tape-membrane*. Notice that the l th tape-membrane has label $b(l)$, if $l \leq n$, and it has label $b(n+1)$ otherwise (we distinguish the indexes of the first $n+1$ tape-membranes in order to ensure that the objects in the input multiset are able to find their corresponding tape-membranes).

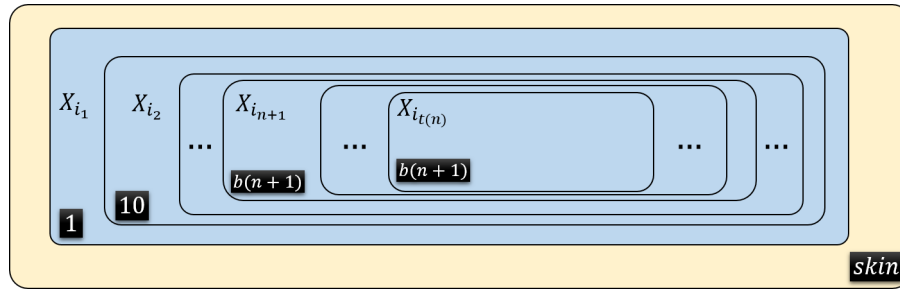


Fig. 1. The membrane structure corresponding to the simulated tape

For every $l \in [t(n)]$, the l th tape-membrane contains further membranes: for every state s_i , it contains $t(n)$ copies of a membrane with label s_i . Such a membrane contains a further elementary membrane with label s'_i . Moreover, the l th tape-membrane contains a symbol $X_j \in \Gamma$ if and only if the l th cell of M

contains the symbol X_j . Furthermore, if M is in state s_i and the head of M points to the l th cell, then an object \uparrow_i is placed into the l th tape-membrane to represent this information (see Fig. 2 where we depicted the third tape-membrane).

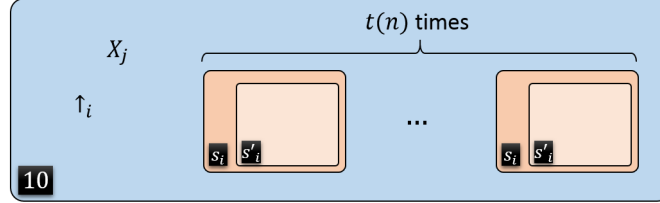


Fig. 2. The membrane structure of the third tape-membrane

We will see that when \uparrow_i and X_j appear in the l th tape-membrane, then these objects will dissolve a membrane pair $[[]_{s'_i}]_{s_i}$ and introduce new objects corresponding to the value $\delta(s_j, X_i)$. With these new objects $II_M(n)$ will be able to maintain its configurations so that finally its current configuration corresponds to the new configuration of M .

The formal definition of $II_M(n)$ is as follows. For every $n \in \mathbb{N}$, let $II_M(n) := (\Sigma', \Gamma', H, \mu, W, R)$, where:

- $\Sigma' := \{a_{b(i)} \mid a \in \Sigma, 1 \leq i \leq n\}$
- $\Gamma' := \Sigma' \cup \Gamma \cup \{\uparrow_i, \downarrow_{i,d} \mid 1 \leq i \leq m, d \in \{L, R\}\} \cup \{d_{b(0)}, \dots, d_{b(2n)}\}$
- $H := \{skin, b(1), \dots, b(n+1)\} \cup \{s_1, \dots, s_m, s'_1, \dots, s'_m\}$;
- μ is a nested membrane structure $[[[\dots[\dots[]_{b(n+1)} \dots]_{b(n+1)} \dots]_{b(2)}]_{b(1)}]_{skin}$ (containing $t(n) - n$ membranes with label $b(n+1)$), such that each membrane in this structure contains a further membrane structure ν , where ν consists of $t(n)$ copies of the membrane structure $[[]_{s'_i}]_{s_i}$, for every $i \in [m]$. The input membrane is $[]_{skin}$;
- $W := w_{skin}, w_{b(1)}, \dots, w_{b(n+1)}, w_{s_1}, \dots, w_{s_m}, w_{s'_1}, \dots, w_{s'_m}$, where
 - $w_{skin} := \varepsilon$;
 - $w_{b(1)} := d_0, w_{b(l)} := \varepsilon$, for every $l \in [2, n]$, and $w_{b(n+1)} := X_k$ (i.e., $w_{b(n+1)}$ is the blank symbol);
 - $w_{s_i} = w_{s'_i} := \varepsilon$, for every $i \in [m]$;
- R is the set of the following rules:
 - Rules to set up the initial configuration of M :
 - (a) $a_{b(i)}[]_{b(l)} \rightarrow [a_{b(i)}]_{b(l)}$, $a_{b(i)}[]_{b(i)} \rightarrow [a]_{b(i)}$, for every $i \in [1, n]$ and $l < i$;
 - (b) $[d_{b(i)} \rightarrow d_{b(i+1)}]_1$, $[d_{b(2n)} \rightarrow \uparrow_1]_1$, for every $i \in [2n - 1]$;
 - Rules for simulating a configuration step of M :
 - (c) $\uparrow_i []_{s_i} \rightarrow [\uparrow_i]_{s_i}$, $[\uparrow_i]_{s_i} \rightarrow \varepsilon$, for every $i \in [1, m]$;
 - (d) $X_j []_{s'_i} \rightarrow [X_j]_{s'_i}$, $[X_j]_{s'_i} \rightarrow X_r \downarrow_{t,d}$, for every $i \in [1, m]$, $j \in [1, k]$ and $(X_r, s_t, d) = \delta(s_i, X_j)$;

- (e) $\downarrow_{i,R} []_{b(l)} \rightarrow [\uparrow_i]_{b(l)}$, for every $i \in [1, m]$, $l \in [1, n + 1]$;
- (f) $[\downarrow_{i,L}]_{b(l)} \rightarrow []_{b(l)} \uparrow_i$, $[\downarrow_{i,L}]_1 \rightarrow [\uparrow_i]_1$, for every $i \in [1, m]$, $l \in [2, n + 1]$;
- Rules for sending out the computed answer to the environment:
 - (g) $[\downarrow_{m-1,d \rightarrow yes}]_{b(l)}$, $[\downarrow_{m,d \rightarrow no}]_{b(l)}$, for every $l \in [1, n + 1]$ and $d \in \{L, R\}$;
 - (h) $[yes]_{b(l)} \rightarrow []_{b(l)} yes$, $[no]_{b(l)} \rightarrow []_{b(l)} no$, for every $l \in [1, n + 1]$;
 - (i) $[yes]_{skin} \rightarrow []_{skin} yes$, $[no]_{skin} \rightarrow []_{skin} no$.

Next, we describe how $\Pi_M(n)$ simulates the work of M . We will see that $\Pi_M(n)$ can set up the initial configuration of M in $O(n)$ steps and that every configuration step of M can be simulated by the P system performing a constant number of steps. We distinguish the following three main stages of the simulation:

Stage 1: Setting up the initial configuration of M . Assume that M is provided with the input word $a_1 a_2 \dots a_n$ ($a_i \in \Sigma, i \in [n]$). Then the input multiset of $\Pi_M(n)$ is $pos(w)$. During the first $2n$ steps, every object $a_{b(i)}$ in the input multiset finds its corresponding membrane with label $b(i)$. At the last step, $a_{b(i)}$ evolves to a , as the sub-index $b(i)$ is not needed any more. Meanwhile, in membrane 1 object d_0 evolves to object $d_{b(2n)}$ and $d_{b(2n)}$ evolves to \uparrow_1 . After these steps, the l th tape-membrane of the system contains an object $X \in \Gamma$ if and only if the l th cell of the tape of M contains X . Moreover, the object \uparrow_1 occurring in the first tape-membrane represents that M 's current state is s_1 (that is, the initial state) and that the head of M points to the first cell. Thus, after $2n$ steps the configuration of $\Pi_M(n)$ corresponds to the initial configuration of M .

Stage 2: Simulating a configuration step of M . Let us assume that M has the configuration appearing in Fig. 3. We can assume that $\Pi_M(n)$ has the corresponding configuration depicted in Fig. 4 (for the sake of simplicity we assume that $l \in [n + 1]$).

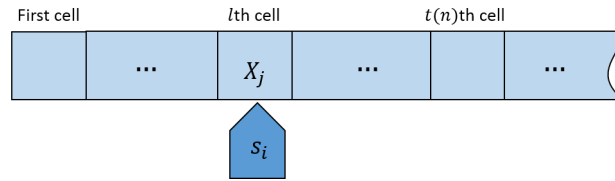


Fig. 3. A configuration of M

The simulation of the computation step of M starts as follows. Firstly, \uparrow_i goes into a membrane with label s_i and then dissolves it using rules in (c). Meanwhile, \uparrow_i evolves to ε . Let us remark that the system always can find a membrane with label s_i in the corresponding tape-membrane. Indeed, at the beginning of the

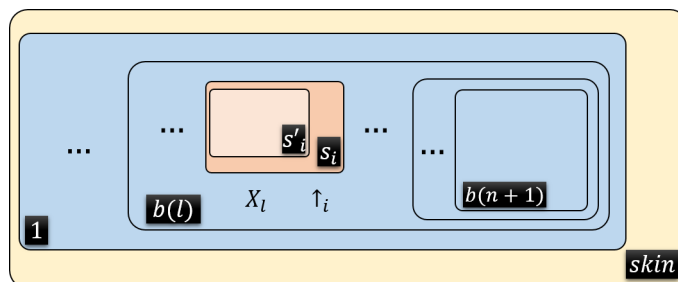


Fig. 4. The corresponding configuration of $\Pi_M(n)$

computation, every tape-membrane contains $t(n)$ copies of a membrane with label s_i . Moreover, M can perform at most $t(n)$ steps and the simulation of one step dissolves exactly one membrane with label s_i .

Next, X_j goes into the membrane s'_i and then dissolves it. During the dissolution two new objects, X_r and $\downarrow_{t,d}$ are introduced according to the value $\delta(s_i, X_j)$. Notice that in $\downarrow_{t,d}$, the index t corresponds to the index of the new state of M and d denotes the direction of the tape head. Now the simulation of the corresponding movement of the head is done as follows. According to the value of d we distinguish the following cases:

Case 1: $d = R$. In this case $\Pi_M(n)$ applies rules in (e): $\downarrow_{t,R}$ is sent into the next inner tape-membrane and, meanwhile, it evolves to \uparrow_t . This corresponds to the move of the tape head to the right.

Case 2: $d = L$. This case is similar to the previous one, but here $\Pi_M(n)$ applies rules in (f): $\downarrow_{t,L}$ is sent out of the current tape-membrane and it evolves to \uparrow_t . This corresponds to the move of the tape head to the left. Notice that if $l = 1$, then $\Pi_M(n)$ can apply only the second rule in (f) which means that in this case \uparrow_t remains in the first tape-membrane. This still corresponds to the step of M , since in this case the head of M cannot move left.

Stage 3: Sending to the environment the correct answer. Whenever an object $\downarrow_{m-1,d}$ ($d \in \{L, R\}$) is introduced in a tape-membrane (i.e., when the simulated M enters its accepting state), the system introduces object *yes* using the first rule in (g). Then this object is sent out of the tape-membranes until it reaches the skin membrane using rules in (h). Finally, *yes* is sent out to the environment using the first rule in (i). $\Pi_M(n)$ performs a similar computation concerning object *no*.

It can be seen using the notes above that $\Pi_M(n)$ is a confluent polarization-less recognizer P system that simulates M correctly. It is also clear that $\Pi_M(n)$ does not employ dissolution and membrane division rules. The other properties of $\Pi_M(n)$ mentioned in Theorem 1 are discussed next.

Time and space complexity of $\Pi_M(n)$. The time complexity of $\Pi_M(n)$ is measured as follows. As we already discussed, *Stage 1* takes $O(n)$ steps. It can be seen that the simulation of a step of M takes five steps. Thus, *Stage 2* takes $O(t(n))$ steps. Finally, *Stage 3* takes also $O(t(n))$ steps. Thus, for every word $x \in \Sigma^*$ with length n , starting $\Pi_M(n)$ with $pos(x)$ in its input membrane, it halts in $O(t(n))$ steps.

Concerning the space complexity, a configuration of $\Pi_M(n)$ contains $t(n)$ tape-membranes and every tape membrane contains $t(n)$ copies of the membrane structure $[[]_{s'_i}]_{s_i}$, for every $i \in [m]$. Moreover, every cell of $\Pi_M(n)$ contains a constant number of objects. Thus, the space complexity of $\Pi_M(n)$ is $O(t^2(n))$.

(DLOGTIME, SPACE($\log t(n)$))-uniformity. We have already discussed that pos , which is the function that we used to encode the input words in Σ^* is in **DLOGTIME**. Thus, it remains to describe a deterministic Turing machine F that can construct $\Pi_M(n)$ using space $O(\log t(n))$. It is clear that the objects in Γ' and the rules of $\Pi_M(n)$ can be enumerated by F using $O(\log n)$ cells. Indeed, Γ' contains $O(mn)$ objects, but m here is a constant that depends only on M . Moreover, $\Pi_M(n)$ has $O(n)$ different rules.

Furthermore, as $\log t(n)$ is space constructable, F can construct $\log t(n)$ in unary form using space $O(\log t(n))$. Using the unary representation of $\log t(n)$, the initial membrane structure of $\Pi_M(n)$ can be constructed by F as follows: when F constructs the l th tape-membrane, then it stores $b(l)$ on one of its tapes using at most $\log t(n)$ cells. Furthermore, when F constructs the k th membrane structure of the form $[[]_{s'_i}]_{s_i}$ ($k \in [t(n)], i \in [m]$) in the l th tape-membrane, then it stores the words $b(k)$ and $b(i)$ on one of its tapes. This also needs $O(\log t(n))$ cells. Thus, the total number of cells used on the work tapes of F when it constructs $\Pi_M(n)$ is $O(\log t(n))$.

4 Conclusions

The simulation of the behaviour of a device of a computation model in a different model allows to see all problems from a new point of view. One of the frontiers of the current research in Membrane Computing corresponds to the computational power of P systems according to the power of the function that encodes the input and the function that constructs the family of P systems.

In this paper, we prove a general result in this line, since we show that every single-tape deterministic Turing machine working in time $t(n)$ can be simulated by a uniform family of recognizer polarizationless P systems with active membranes. Moreover, this is done without significant slowdown in the working time. Furthermore, if $\log t(n)$ is space constructible, then the members of the family can be constructed by a family machine that uses space $O(\log t(n))$.

As it is pointed out in [2], uniform families of polarizationless P systems with active membranes and without dissolution rules are at most as powerful as the used input encoding function (see Theorem 10 in [2]). This fact, together with the result of this paper, illustrates the importance of dissolution rules in

P systems with active membranes when the polarizations of the membranes are not allowed.

As it is mentioned above, if the simulated Turing machine M works in time $t(n)$ and $\log t(n)$ is space constructible, then Π_M can be constructed using space $\log t(n)$. As a particular case, it means that if $t(n)$ is a polynomial function, then Π_M is a **(DLOGTIME, L)**-uniform family of P systems. Likewise, if $t(n)$ is an exponential function, then Π_M is a **(DLOGTIME, PSPACE)**-uniform family.

It remains as an open question if **(DLOGTIME, SPACE($\log t(n)$))**-uniformity in Theorem 1 can be strengthened to **(DLOGTIME, L)**-uniformity (i.e., whether the construction of Π_M can be done using logarithmic space whatever is the running time of M). In our construction membrane division rules are not employed. Nevertheless, even if we used these rules, it is not clear how the tape-membranes or the membrane structures in them could be constructed using logarithmic space. This might be a subject of further research.

Acknowledgements

The authors gratefully acknowledges the helpful suggestions and comments of the anonymous referees. Miguel A. Gutiérrez-Naranjo acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. *Theoretical Computer Science* 529(0), 69 – 81 (2014)
2. Murphy, N., Woods, D.: The computational complexity of uniformity and semi-uniformity in membrane systems. In: Martínez-del-Amor, M.A., Orejuela-Pinedo, E.F., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) *Seventh Brainstorming Week on Membrane Computing*. vol. II, pp. 73–84. Fénix Editora, Sevilla, Spain (2009)
3. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun et al. [10], pp. 302 – 336
4. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. In: Csuhaj-Varjú, E., Kintala, C., Wotschke, D., Vaszil, G. (eds.) *Proceeding of the 5th Workshop on Descriptive Complexity of Formal Systems*. DCFS 2003, pp. 284–294 (2003)
5. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics* 11(4), 423–434 (2006)
6. Porreca, A., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. *International Journal of Computers, Communications and Control* 4(3), 301–310 (2009)
7. Porreca, A.E.: *Computational Complexity Classes for Membrane System*. Master's thesis, Università di Milano-Bicocca, Italy (2008)

8. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, Gy. (eds.) International Conference on Membrane Computing. Lecture Notes in Computer Science, vol. 7762, pp. 342–357. Springer (2012)
9. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics* 6(1), 75–90 (2001)
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England (2010)
11. Romero Jiménez, Á., Pérez-Jiménez, M.J.: Simulating Turing machines by P systems with external output. *Fundamenta Informaticae* 49(1-3), 273–278 (2002)
12. Sipser, M.: *Introduction to the Theory of Computation*. Cengage Learning (2012)
13. Valsecchi, A., Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) *Workshop on Membrane Computing*. Lecture Notes in Computer Science, vol. 5957, pp. 461–478. Springer, Berlin Heidelberg (2009)

Categorised Counting Mediated by Blotting Membrane Systems for Particle-based Data Mining and Numerical Algorithms

Thomas Hinze^{1,2}, Konrad Grützmann³, Benny Höckner¹
Peter Sauer¹, and Sikander Hayat⁴

¹Brandenburg University of Technology
Institute of Computer Science and Information and Media Technology
Postfach 10 13 44, D-03013 Cottbus, Germany

²Friedrich Schiller University Jena
Ernst-Abbe-Platz 1–4, D-07743 Jena, Germany

³Helmholtz Centre for Environmental Research – UFZ
Permoserstr. 15, D-04318 Leipzig, Germany

⁴Harvard Medical School, 200 Longwood Avenue, Boston, MA 02115, USA

{thomas.hinze,benny.hoeckner,peter.sauer}@tu-cottbus.de,
konrad.gruetzmann@ufz.de, Sikander.Hayat@hms.harvard.edu

Abstract. Blotting turns out to be a rather common and effective approach in molecular information processing. An initial pool of molecules considered as sets of individual data becomes spatially separated according to the presence or absence of specific attributes like weight index or chemical groups and labels. In this connection, molecules with similar properties form a spot or blot. Finally, each blot can be visualised or analysed revealing a corresponding score index or count from the number of accumulated molecules. The entire variety of blots emerged over time provides crucial and condensed information about the molecular system under study. Inspired by the idea to obtain a significant data reduction while keeping the essential characteristics of the molecular system as output, we introduce blotting membrane systems as a modelling framework open for numerous applications in data mining. By means of three dedicated case studies, we demonstrate its descriptive capability from an explorative point of view. Our case studies address particle-based numerical integration which suggests a model for the synchronised 17-year life cycle of *Magicalicadas*. Furthermore, we exemplify electrophoresis to carry out a variant of bucket sort.

1 Introduction and Background

From a technical point of view, biological information processing as well as molecular computing and particle-based algorithms commonly result in a huge amount of more or less raw data. Afterwards, an appropriate interpretation of these data

sets from a holistic perspective towards evident conclusions turns out to be a challenging task [14]. In many cases, this is due to the partly diffuse nature of molecular processes at a *mesoscale* level. By means of this term, we summarise a typical outcome of a measure or a direct observation obtained from an underlying experiment or system under study. Interestingly, the majority of visualisation techniques and analytical tools in biochemistry comprises a *spatial separation* of molecules or particles based on relevant attributes or properties. For instance, an electrophoresis spreads electrically charged molecules by their individual weights [15]. Other attempts to capture a molecular system employ fluorescence labels specifically attached to signalling molecules [4, 16]. Here, the spatial distribution of these labels represents the overall behaviour of the system. In addition, high-resolution microscopy in many facets sheds light on the geometrical location of molecular clusters or even single molecules up to a nanometre scale [5].

All these techniques have in common that the number of molecules or particles present within predefined grid *regions* or accumulated towards distinguishable *clusters* gives the crucial information about the underlying system. Often, the number of molecules needs to overcome a certain *threshold* before taken into consideration for detectability. Finally, the number of molecules subject to a geometrical grid tends to be expressed by a kind of heatmap. Here, the range of potential molecular amounts becomes divided into several disjoint intervals while each of them is assigned to a corresponding colour or intensity value. Having a look at the grid mostly reveals a collection of spots or *blots* in different size and colour, sometimes partially overlapping, and almost certainly somehow blurry or frazzled.

A variety of dedicated blotting techniques which emerged during the last decades produces two-dimensional blot diagrams, sometimes also called *blotting papers* [3, 24]. The metaphor rather illustratively describes this form of data representation, especially in case of prevalent Southern, Western, and Northern blots [22]. These techniques allow among others a spatial separation of DNA (deoxyribonucleic acid), RNA (ribonucleic acid), or labelled proteins from an initial mixture according to their weights or according to the presence or absence of oligomeric subsequences. In application scenarios with medical background, resulting blot diagrams help to identify whether or not certain viral infections or gene mutations occurred. Both reasons can imply synthesis of malformed, functionally insufficient proteins [26]. From the percentage of malformed proteins in comparison to the total amount, a stage or degree of the damage can be hypothesised if there is enough empirical, statistical, or deducible significance.

The effect of spatial blotting can also be seen in nature. Control of cell differentiation and proliferation in the embryo of the fruit fly *Drosophila melanogaster* gives a fascinating example [25]. The underlying processes manage the formation of embryonic patterns from the fertilised egg [23]. Embryonic patterns mark an essential intermediate state in the morphogenesis of the organism which leads to its functional structures and appendages. Beyond the body proportions, also shape, positions and size of head and thorax and more fine-grained, those of organs like eyes and wings during maturation become mainly determined by

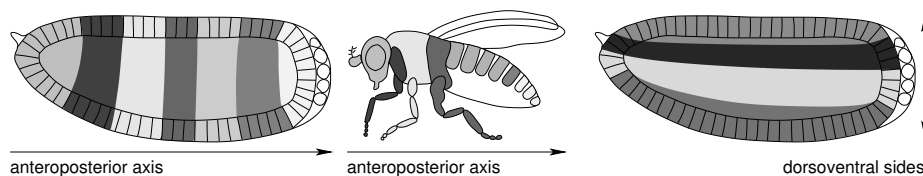


Fig. 1. Schematic representation of an embryonic pattern in the fertilised egg of the fruit fly *Drosophila melanogaster*. The pattern forms a 7×4 -grid whose 28 coordinated regions are characterised by individual presence of specific cytokine combinations. Along with cell differentiation and proliferation during maturation, functional structures and appendages of the organism emerge from corresponding initial regions of the embryonic pattern. It constitutes the anteroposterior axis together with its dorsoventral sides as well. Graphical representation inspired by fluorescence microscopy images (Hox coordinate gene expression states) in [25].

embryonic patterns. In simple terms, the embryonic pattern quite precisely defines a geometrical grid throughout the longitudinal (anteroposterior) axis and orthogonally along the so-called dorsoventral sides, see Figure 1. In the first developmental stage, the embryonic pattern separates seven dedicated regions anterioposterially and four regions dorsoventrally. Each of these regions is characterised by a specific mixture of a number of proteins called *cytokines* whose individual concentrations are spatially distributed according to the regions. Typically, within a region a total amount of one up to three different cytokines exhibits a high concentration while all other cytokines persist in low concentrations. The grid-like spatial distribution of cytokines results from an underlying expression scheme of so-called coordinate genes. The activation of these genes is controlled by a reaction cascade in several phases along a predefined time course. The activation cascade starts simultaneously from both opposite tips of the embryo triggered by the zygote (fertilised egg cell) in concert with environmental stimuli. In terms of a systemic understanding, the process of embryonic pattern formation reflects the functionality of *categorised counting*. Here, the categories symbolise all single regions from which the grid is composed of, while the ratios of individual cytokine concentrations accumulated within each region indicate its spatial position inside the entire grid. In other words, each region can be identified by a specific index (count value) estimated from the involved cytokine concentration gradients. In subsequent phases of maturation, the initial embryonic pattern becomes more and more refined. To do so, the cytokines act as transcription factors specifically activating those parts of the genome whose resulting proteins lead to differentiated cells forming the variety of body structures.

All scenarios addressed here so far share a common essential principle consisting of three consecutive steps:

1. Particles or molecules which represent a pool of data create spatially distributed blots of diverse size or density. In a general term, this step is called *clustering*. Initiated by a more or less complex interplay of physical processes

like transduction or diffusion mainly in conjunction with (bio)chemical reactions, clustering follows known principles of natural laws but can be enriched by stochastic effects. This stochasticity causes a certain bias inflating the amount of raw data. In order to enable a later analysis, the geometrical location of each cluster needs to be captured in an appropriate manner. Most simply, the boundaries of each cluster might be marked a priori. In case the expected cluster positions remain unknown until the clustering is completed, a subsequent *classification* turns out to be the method of choice instead, especially if there is evidence of (partially) overlapping blots.

2. The number of relevant molecules within each cluster defines a dedicated qualitative score like a colour intensity or just an index for instance. This step can be covered by the term *counting*. The ability to carry out counting comes along with a previously finalised *categorisation*. To this end, a predefined setting of available categories becomes identified which in turn specifies the domain of potential and acceptable scores. A final set of distinct greyscales or a range of discrete measurement readings give typical examples for categories. Eventually, counting maps the molecular abundance into the corresponding category. Hence, we obtain a more or less tremendous data reduction. Afterwards, each cluster exhibits its count as essential information.
3. Based on the clusters together with its corresponding counts, the final step comprises the generation of system's *response*. Living organisms taken as natural systems like the fruit fly exhibit a special behavioural pattern like maturation for response. In contrast, man-made systems like electrophoresis chambers request an external mathematical analysis instead, mostly statistics or deduction. Anyway, a computation typically takes place having the response as output.

We say that a system operating in this manner performs *categorised counting*. Inspired by numerous further examples, the idea arises whether this common behavioural principle can be advantageously captured by a consistent dedicated description framework. Therefore, the notion of a *P system* [20] seems to be an ideal candidate due to its multiset-based nature. A *multiset* inherently supports the strategy of categorised counting: Elements reflect the categories while its multiplicities stand for the corresponding count. Clustering is expressed by accumulation of elements into the underlying multiset. Putting all necessary descriptive ingredients together will lead us to the introduction of *blotting membrane systems*, a new class of P systems aimed at formalisation of categorised counting. The level of abstraction taken into consideration should be balanced in order to cope with the computational complexity of multiple particle systems entering the clustering process. The main focus is laid to facilitate explorative in-silico studies able to extract a condensed but sufficient description of the systems behaviour and/or possible conclusions from a (huge) set of slightly biased raw data. Having in mind that particularly the response stage in categorised counting might incorporate statistical techniques and deductive methods applied to an initial set of data, blotting membrane systems can open the field of

data mining [8, 10] for membrane computing. To our best knowledge, this is the first attempt to primarily addressing this line of research.

In Section 2, we familiarise the reader with the formal definition of blotting membrane systems together with all required prerequisites. Hereafter, three case studies selected from different facets of molecular information processing demonstrate the descriptive capability. We start with an initial example, particle-based numerical integration, presented in Section 3. This feature in addition with a limiting threshold is sufficient for a simple behavioural model on how insects of the species *Magicicada* can count a time span of rather exactly 17 years which comprises the synchronised life cycle of all individuals within the population. Section 4 is dedicated to electrophoresis, a commonly used technique to arrange electrically charged molecules by their weights. The resulting systems behaviour resembles the algorithmic strategy of bucket sort. A final discussion concludes benefits as well as open questions for future work.

2 Blotting Membrane Systems

Formal Prerequisites

Let A and B be arbitrary sets, \emptyset the empty set, \mathbb{N} the set of natural numbers including zero, \mathbb{R} the set of real numbers, and \mathbb{R}_+ the set of non-negative real numbers. A and B are disjoint (share no common elements) iff $A \cap B = \emptyset$. The Cartesian product $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$ collects all tuples from A and B . For $A \times A$, we write A^2 for short. The term $\text{card}(A)$, also written as $|A|$, denotes the number of elements in A (cardinality). A multiset over A is a mapping $F : A \rightarrow \mathbb{N} \cup \{+\infty\}$. Multisets in general can be written as an elementwise enumeration of the form $\{(a_1, F(a_1)), (a_2, F(a_2)), \dots\}$ since $\forall (a, b_1), (a, b_2) \in F : b_1 = b_2$. The support $\text{supp}(F) \subseteq A$ of F is defined by $\text{supp}(F) = \{a \in A \mid F(a) > 0\}$. A multiset F over A is said to be empty iff $\forall a \in A : F(a) = 0$. The cardinality $|F|$ of F over A is $|F| = \sum_{a \in A} F(a)$.

Definition of Systems Components

A *blotting membrane system* is a construct

$$\boxed{II} = (P, L, C, B_1, \dots, B_{|C|}, S, R, r) \quad (1)$$

whose components mimic all ingredients of categorised counting, namely

- spatially distributed particles equipped with labels,
- resulting blots acting as categories,
- score values (counts) according to each category, and
- a response as follows:

We assume that the systems input consists of a final set of particles arbitrarily located at a two-dimensional, not necessarily bounded grid. Therefore, each particle carries its grid coordinates (Cartesian or by generic system) together with an individual label (like weight index or presence of a chemical fluorescence marker) beforehand responsible for processing the spatial separation.

L	arbitrary set of available labels
$P \subset \mathbb{R} \times \mathbb{R} \times L$	final set of particles, each of them specified by grid position and label

By means of the components C and $B_1, \dots, B_{|C|}$, we capture the arrangement of particles into blots as well as the assignment of categories to the blots.

C	arbitrary set of available categories either defined explicitly or obtained implicitly as result of a classification over P
$B_1 \subseteq P$ \vdots $B_{ C } \subseteq P$	entirety of blots, each of them specified by the accumulated particles

In simple cases, the number of categories $|C|$ can be set explicitly, particularly in those studies which enable a precise prediction of the expected blots due to a sufficient knowledge of the systems behaviour. More elaboratively, the number of categories can result from a classification process taking into account spatial distances (or distance measures) between all or selected particles. To this end, a distance matrix of the incorporated particles is created first making each particle its own singleton category for the beginning. Out of the matrix, those two particles exhibiting the minimal distance to each other become identified. They are removed from the matrix by being merged into a common category. This new category is placed in the distance matrix again. This comes along with filling its distances to all other categories present in the matrix. There are different calculation schemes on how to obtain these distance updates originating a variety of classification approaches [6]. By iterating the merge of two nearest categories, the total number of categories decreases more and more. Usually, the merging process stops after a certain distance threshold is reached or a desired final number of categories emerged (*agglomerative hierarchical clustering*, [11]).

We denote the blots by the family of sets B_1 until $B_{|C|}$. If the condition $\bigcap_{i=1}^{|C|} B_i = \emptyset$ holds, the blots are said to be *non-overlapping*. Please note that the blots do not need to be necessarily disjoint. In general, a particle is allowed to be part of several (overlapping) blots.

$S : C \rightarrow \mathbb{N}$	multiset subsuming the score values (counts) over all categories
R	arbitrary set specifying the response domain
$r : \mathbb{N}^{ C } \rightarrow R$	response function

The multiset S appears to represent the central part of the blotting membrane system since its core functionality of data reduction by categorised counting is expressed here. Typically, we choose $S(c) = |B_c|$ for all $c \in C$. In order to finalise

systems description, we still need to formalise its response. In this context, we initiate a response domain R capturing the whole of potential systems outputs. Based on that, the response function r analyses the counts of all categories. Finally, it derives the corresponding response. This step might include statistical tests and/or some dedicated reasoning. Hence, the formal description of function r within application scenarios might become rather extensive.

A Toy Example: Particle-based Approximation of Constant $\pi \approx 3.14$

Using a toy example, we illustrate the formalism of blotting membrane systems. To this end, we exemplify a simple particle-based rational approximation of the mathematical constant π . This application scenario utilises a square-shaped underlying grid equipped with Cartesian coordinates along with a centered point of origin. The grid inscribes a circle, for simplicity we choose a radius whose unit of length equals to 1. Now, a huge number of particles is randomly distributed on the underlying grid taking care that an equipartition of the particles is met. The equipartition ensures a spatial homogeneity. Those particles placed within the circle form a blot with a corresponding category on its own, see Figure 2. Beyond that, all particles in the whole grid are considered as a (partially overlapping) blot as well substantiating a second category. Towards an approximation of $\pi = 3.14159265\dots$, we only need to count the number of particles in both categories. Since the circle with radius 1 covers an area of π while the grid constitutes 4 surface units. As an approximation, we end in the responding relation:

$$\frac{\pi}{4} = \frac{\text{number of particles placed within the circle}}{\text{number of particles in total on the whole grid}}$$

Let us now formulate the dedicated blotting membrane system. To do so, we assume to have enough previously scattered particles P at hand whose coordinates come either from a numerical simulation based on a (pseudo) random number generator or from direct physical measurement. We employ a uniform label l for all particles. A resulting system might read:

$$\boxed{\Pi} = (P, L, C, B_1, \dots, B_{|C|}, S, R, r) \quad \text{with} \\ L = \{l\}$$

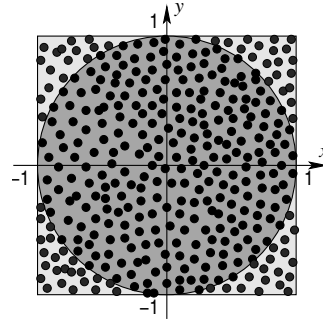


Fig. 2. Particles randomly distributed in spatial equipartition on an underlying square-shaped grid. The number of particles placed within the inscribed circle in comparison to the total number of particles provides a rational approximation of the constant π .

$$\begin{aligned}
P &= \{(0.70191, -0.21355, l), (0.02273, 0.91508, l), \dots, (-0.45160, 0.52241, l)\} \\
C &= \{\odot, \square\} \\
B_{\odot} &= \{(x, y, l) \mid (x, y, l) \in P \wedge x^2 + y^2 \leq 1\} \\
B_{\square} &= \{(x, y, l) \mid (x, y, l) \in P \wedge |x| \leq 1 \wedge |y| \leq 1\} \\
S(c) &= |B_c| \quad \forall c \in C \\
R &= \mathbb{R} \\
r(S) &= 4 \cdot \frac{S(\odot)}{S(\square)}
\end{aligned}$$

A simulation case study discloses following numerical results for instance:

	$ P $	$S(\odot)$	$S(\square)$	rational approximation r of π
	10,000	7,928	10,000	<u>3.1712</u> ... (2 reliable digits)
	1,000,000	785,502	1,000,000	<u>3.1421</u> ... (3 reliable digits)
	100,000,000	78,542,447	100,000,000	<u>3.1417</u> ... (4 reliable digits)

Obviously, an ascending number of particles involved in the system leads to a higher accuracy of the approximation. Nevertheless, we are aware of the slow convergence behaviour. An additional decimal digit of π reliably figured out by the blotting membrane system requires a 100-fold increase of the total particle number due to the two-dimensional nature of the experimental setting. Of course, the numerical precision of particle coordinates needs to be adapted as well.

3 Particle-based Numerical Integration

An evident application of categorised counting for molecular computation can be found in particle-based numerical integration. Enhancing the idea of a particle-based rational approximation of the constant π introduced in the previous section, we prepare a two-dimensional grid with the complete course of the desired real-valued function $f : \mathbb{R} \rightarrow \mathbb{R}_+$ to be integrated numerically within a range $[a, b]$ of interest, see Figure 3. Subsequently, a huge number of particles becomes consistently scattered over the whole grid producing a spatially homogeneous particle distribution. Within a separate category, the number of particles placed below the function course of f is achieved by counting. Its amount in comparison with the total number of particles on the whole grid offers a rational approximation of the numerical integral to be calculated:

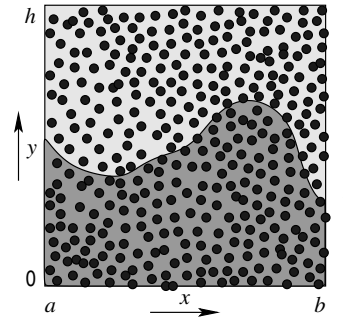


Fig. 3. Operating scheme of a particle-based numerical integrator

$$\frac{\int_a^b f(x) dx}{h \cdot (b - a)} = \frac{\text{number of particles placed below the function course of } f}{\text{number of particles in total on the whole grid}}$$

Written as a blotting membrane system, we obtain for instance (particle coordinates adjusted to $h = 10, a = 0, b = 10$):

$$\begin{aligned} \boxed{\Pi} &= (P, L, C, B_1, \dots, B_{|C|}, S, R, r) \quad \text{with} \\ L &= \{l\} \\ P &= \{(3.46119, 1.83835, l), (0.92240, 2.70318, l), \dots, (4.07919, 3.95624, l)\} \\ C &= \{\int, \square\} \\ B_f &= \{(x, y, l) \mid (x, y, l) \in P \wedge x \geq a \wedge x \leq b \wedge y \geq 0 \wedge y \leq f(x)\} \\ B_\square &= \{(x, y, l) \mid (x, y, l) \in P \wedge x \geq a \wedge x \leq b \wedge y \geq 0 \wedge y \leq h\} \\ S(c) &= |B_c| \quad \forall c \in C \\ R &= \mathbb{R} \\ r(S) &= h \cdot (b - a) \cdot \frac{S(\int)}{S(\square)} \end{aligned}$$

An impressive example for biological exploitation of numerical integration is inspired by cicadas, insects of the species *Magicicada*. Populations in northern America share a synchronous life cycle of 17 years while those in central America prefer 13 years [18]. Most of its existence is spent underground in a dormant state. Shortly before the end of the life cycle, all the adults of a brood emerge at roughly the same time to reproduce for several weeks. After laying their eggs, the adults die off and the cycle begins again. What stands out is that 17 and 13 are prime numbers, which suggests that the reproduction period does not coincide with the life cycles of potential predators. The simultaneous mass awakening of a brood also ensures that predators are overwhelmed by the number of cicadas so that a large number can survive. In order to guarantee a concerted awakening of all members of a brood, the species needs a precise molecular mechanism to measure the passage of the appropriate amount of time. Since it seems that there is no external stimulus with a natural period of 13 or 17 years, its exact estimation exclusively based on annual or even shorter cycles becomes a complicated task [28].

There is some evidence for a potential annual stimulus utilised by periodical cicadas: sap circulating through the root capillars. Its intensity alters between high abundance during the growth period and almost absence during winter [7]. Cicada larvae could make use of the sap for nutrition while metabolic byproducts accumulate in terms of a numerical integration from its temporal course. Following this idea, the byproducts would persist within one or several vesicles whose outer membranes are going to burst after its content has reached a certain mass.

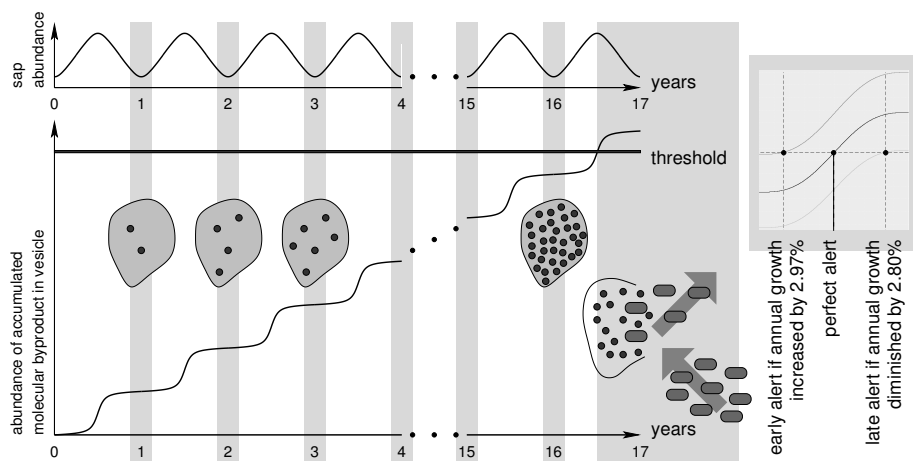


Fig. 4. Schematic representation of a molecular numerical integrator possibly residing in periodical 17-year cicadas for control and synchronisation of population life cycle

Up to now, we failed in retrieving detailed scientific publications on hypothesised or even verified mechanisms. A more or less speculative model aims at a combination of two processes, a slow growth on the one hand and a threshold on the other. Growth means a successive accumulation of a dedicated species. As soon as its concentration exceeds an inherently set threshold, the finalisation of the life cycle is initialised indicating the elapsed amount of 17 years. A successive accumulation organised for instance in annual cycles is useful for a high precision. To this end, a core oscillator (like periodical sap cycle) could provide an annually altering signal of the form $a + \sin(bt)$ subject to time t . A simple signal integration then produces a temporal course of the form $at - \frac{1}{b} \cdot \cos(bt) + C$ with a successive, staircase-shaped growth, see Figure 4.

The molecular alert after reaching the threshold could initiate a signalling cascade which in turn releases trigger molecules into the environment. The trigger molecules on their own could interact by a special form of *quorum sensing* [2, 19]: A cicada larva needs to perceive enough trigger molecules in conjunction with having met its inherent threshold in order to finalise its life cycle. This finetuning synchronisation strategy seems to be sufficiently robust but nevertheless, it is more prone to premature or late alert than a discretely operating n -ary counter which in contrast requires a large and complex reaction network [13]. In simulation studies, we empirically found out that annual variation of byproduct increase within a range of approx. $-2.80\% \dots +2.97\%$ can be tolerated by sinusoidal numerical integration keeping the point in time when the threshold is reached within the growth period of year 17 (Figure 4, right part). Furthermore, a numerical integrator in concert with a finetuning sensing mechanism for population-wide synchronisation is sufficient to toggle the life cycle between a variety of years by a low number of slight evolutionary changes. Hav-

ing this feature at hand, it becomes plausible how a widespread range of life times could emerge where those forming prime numbers resist the evolutionary selection driven by predators.

Let us now formalise the speculative model using a blotting membrane system which is focused on alerting life cycle finalisation. Here, the categorised counting mechanism needs to distinguish the accumulated byproduct (A) on the one hand and the sensed trigger molecules (T) on the other. To this end, we employ two molecular labels A and T , respectively. The underlying grid is symbolised by a plan view into the soil (arbitrarily chosen: $0 \leq x \leq 8$ and $0 \leq y \leq 3$). A spatial cluster of byproducts A (maximum cluster inherent distance: 0.03) marks the position of a cicada larva. Within a local circular environment (maximal distance: 0.8), it detects trigger molecules. Systems response comprises overall alerting on life cycle finalisation:

$$\begin{aligned}
\boxed{\Pi} &= (P, L, C, B_1, \dots, B_{|C|}, S, R, r) \quad \text{with} \\
L &= \{A, T\} \\
P &= \{(1.123, 1.992, A), (1.125, 2.001, A), \dots, (4.338, 0.874, T)\} \\
C &= \cup B_{(\bar{x}, \bar{y}, l)} \quad \text{with } l \in L \\
B_{(\bar{x}, \bar{y}, A)} &= \{(x, y, A) \mid (x, y, A) \in P \wedge \forall (a, b, A) \in P : (x - a)^2 + (y - b)^2 \leq 0.03\} \\
&\quad \text{whereas} \\
\bar{x} &= \frac{1}{|B_{(\bar{x}, \bar{y}, A)}|} \cdot \sum_{(x, y, l) \in B_{(\bar{x}, \bar{y}, A)}} x \quad \text{and} \quad \bar{y} = \frac{1}{|B_{(\bar{x}, \bar{y}, A)}|} \cdot \sum_{(x, y, l) \in B_{(\bar{x}, \bar{y}, A)}} y \\
B_{(\bar{x}, \bar{y}, T)} &= \{(x, y, T) \mid (x, y, T) \in P \wedge (x - \bar{x})^2 + (y - \bar{y})^2 \leq 0.8\} \\
S(\bar{x}, \bar{y}, A) &= |D| \quad \text{with} \\
D &= \{B_{(\bar{x}, \bar{y}, A)} \mid |B_{(\bar{x}, \bar{y}, A)}| \geq \text{threshold}_A \wedge |B_{(\bar{x}, \bar{y}, T)}| \geq \text{threshold}_T\} \\
R &= \mathbb{N} \\
r(S) &= |\text{supp}(S)|
\end{aligned}$$

Figure 5 illustrates the formalism. P is composed of all dark dots, small ones and ellipsoidal ones. Small dots represent accumulated byproducts A arranged in four clusters while ellipsoidal dots exhibit trigger molecules T, all of them spatially distributed on a grid (soil). The classification identifies four byproduct categories corresponding to the clusters. They are named in accordance to their central coordinates: $B_{(1,2,A)}$, $B_{(3,1,A)}$, $B_{(5,1.5,A)}$, $B_{(7,1,A)}$. Each of them stands for a cicada larva surrounded by a circular environment which sensibilises for trigger molecules: $B_{(1,2,T)}$, $B_{(3,1,T)}$, $B_{(5,1.5,T)}$, $B_{(7,1,T)}$. Especially, $B_{(7,1,T)} = \emptyset$ since there are no trigger molecules here. Let us require at least 20 byproduct molecules to form threshold_A and at least 5 trigger molecules for threshold_T . Multiset S decides for each larva whether or not it exceeds both thresholds: $S = \{((1, 2, A), 1), ((3, 1, A), 1), ((5, 1.5, A), 0), ((7, 1, A), 0)\}$. The final response provides the number of larvae alerting finalisation of life cycle, here $r(S) = 2$.

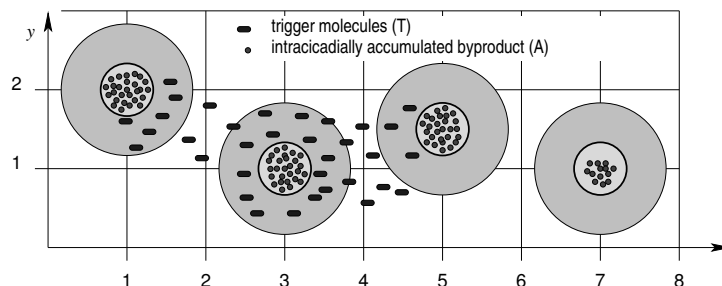


Fig. 5. Fictive cicada population given by a two-dimensional distribution of trigger molecules (T) released by three cicadas and individually accumulated byproduct (A). Central positions of cicadas were chosen randomly, molecules T and A also randomly placed within and around circular regions assigned to each cicada. Trigger molecules T reflect outcome of a potential quorum sensing process at a fixed point in time.

4 Electrophoresis: A Molecular Bucket Sort

Electrophoresis subsumes a physical technique able to spatially separate electrically charged molecules by their weights [15]. Particularly, DNA (negatively charged) and many naturally originated proteins (twisted and folded chains of amino acids whose electrical charge is mainly determined by outer amino acid side chains) are beneficial candidates for widespread applications in molecular biology and chemical analysis [27].

Mostly, electrophoresis takes place within a special physical medium like a *gel* which carries and steers the molecules during the separation process. To do so, the gel is prepared in a way to be equipped with numerous *pores* forming woven channels or tunnels sufficiently sized to allow passage of charged sample molecules. For instance, *agarose* is commonly used to compose a gel suitable for electrophoresis on DNA. The fibre structure of agarose enables pores whose diameter usually varies between 150 and 500 nanometres while a DNA strand (in biologically prevalent B-DNA conformation) diametrically consumes merely 2 nanometres but its length can reach several hundred nanometres [9]. The ready-made gel, typically between 10 and 30 centimetres in length or width and up to 5 millimetres thick, is embedded in a *gel chamber* filled up with a buffer solution in order to adjust an appropriate pH environment. The gel chamber comes with two electrodes, a negative one and a positive one, placed at the opposite boundaries of the gel, see Figure 6.

Subsequently, the sample mixture of DNA strands to be separated becomes injected into the gel close to the negative electrode. Now, an electrical direct-current (DC) voltage, provided by an external power supply and mostly chosen between 80 and 120 volts, is applied to the electrodes. Driven by the electrical force, the negatively charged molecules begin to run towards the positive electrode along a lane through the pores of the gel. In order to mobilise, each molecule has to overcome its friction notable in both forms, with the gel on the one hand and inherently on the other. Interestingly, the resulting velocity of

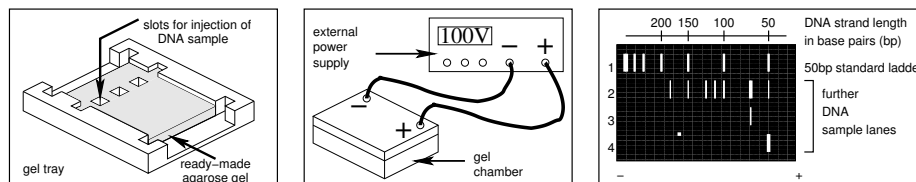


Fig. 6. Sketching technical instruments and outcome of agarose gel electrophoresis.

movement strongly depends on the mass (weight) of the individual molecules. Since small and light molecules induce a low friction, they run faster than heavier exemplars. This distinction finally effects the resulting spatial separation according to the weights of involved charged molecules. The process of electrophoresis is stopped by switching off the voltage shortly before the smallest molecules have reached the opposite end of the gel. For an easier visualisation of this process, the molecular mixture initially becomes enriched by a weakly binding dye whose velocity converges in compliance with the smallest sample molecules [27].

In addition, the DNA sample molecules had been stained using a fluorescence marker like *ethidium bromide* [21]. This substance loosely binds to the hydrogen bonds of double-stranded DNA and persists at the DNA during the electrophoresis run. Ethidium bromide attached to DNA fluoresces under ultra violet (UV) light making the DNA visible inside the gel. Typically, the DNA after electrophoresis is arranged in so-called *bands* (sustained bar-shaped blots) along the underlying lane. Normally, these bands appear in light-grey up to white colours on a dark gel background. The colours intensity gives a raw information on the absolute number of molecules of almost the same mass accumulated within each band, see Figure 7, left part. In a first and mostly sufficient approximation, gel electrophoresis can be modelled by an obvious equation. The electrical force F_E needs to overcome the friction F_R . Movement of charged molecules starts up iff both forces are almost in parity to each other with slight (negligible) emphasis on F_E :

$$F_E \geq F_R, \text{ in good approximation } F_E = F_R$$

Now, we can resolve both forces by formulating its strength using a couple of dedicated parameters. The electrical force is defined as the product of the molecular electrical charge q with the electrical field E which in turn can be measurably expressed by the quotient of the voltage U and the distance h between the electrodes: $F_E = q \cdot E = q \cdot \frac{U}{h}$. In contrast, the friction in accordance with *Stokes' law* reads: $F_R = 6 \cdot \pi \cdot \eta \cdot r \cdot v$ assuming movement of a sphere where r denotes the radius, v symbolises its velocity, and η stands for the viscosity of the medium, mainly reflecting the average size of the pores. The velocity can be assumed to remain almost constant after a short acceleration phase in conjunction with switching on the electrical voltage. Putting everything together reveals:

$$v = \frac{q \cdot E}{6 \cdot \pi \cdot \eta \cdot r}$$

The only indetermined parameter is the radius r of the anticipated sphere representing the moving charged molecule. In order to cope with that, we can imagine that the volume V_{molecule} of the charged molecule resembles the volume V_{sphere} of the anticipated sphere. Having this in mind, we can write $V_{\text{molecule}} = \frac{m}{\rho}$ with m denoting the mass (weight) of the molecule and ρ its density. Moreover, $V_{\text{sphere}} = \frac{4}{3} \cdot \pi \cdot r^3$. From that, we obtain:

$$r = \left(\frac{3}{4 \cdot \pi} \cdot \frac{m}{\rho} \right)^{\frac{1}{3}}$$

Let us now compose a resulting function $s : \mathbb{R}^2 \rightarrow \mathbb{R}$ which describes the distance moved by a charged molecule with mass m after an elapsed time t :

$$\begin{aligned} s(m, t) &= v \cdot t \\ &= \frac{q \cdot E}{6 \cdot \pi \cdot \eta \left(\frac{3 \cdot m}{4 \cdot \pi \cdot \rho} \right)^{\frac{1}{3}}} \cdot t \\ &= \underbrace{\frac{q}{6 \cdot \pi \cdot \left(\frac{3}{4 \cdot \pi \cdot \rho} \right)^{\frac{1}{3}}}}_{\text{taken as global parameter } G} \cdot \frac{E}{\eta} \cdot \frac{1}{m^{\frac{1}{3}}} \cdot t \\ &= G \cdot \frac{E}{\eta} \cdot \frac{1}{m^{\frac{1}{3}}} \cdot t \end{aligned}$$

For DNA agarose gel electrophoresis, the electrical field E frequently constitutes between $400 \frac{\text{V}}{\text{m}}$ and $500 \frac{\text{V}}{\text{m}}$ while the viscosity commonly differs from $0.001 \frac{\text{kg}}{\text{m} \cdot \text{s}}$ (consistency like water in large-pored gels) up to $0.02 \frac{\text{kg}}{\text{m} \cdot \text{s}}$ in small-meshed gels enhancing the friction along with producing heat. From empirical studies, we fitted a constant average value of approx. $6.794 \cdot 10^{-4} \frac{\text{A} \cdot \text{s} \cdot \text{kg}^{\frac{1}{3}}}{\text{m}}$ for G in agarose gel electrophoresis on double-stranded non-denaturing DNA. When employing the molecule mass m in kg along with elapsed time t in s and remembering that $1\text{VAs} = 1 \frac{\text{kg} \cdot \text{m}^2}{\text{s}^3}$, the final value of the function is returned in metres.

By means of a complementary study, we face the mathematical model with corresponding experimental data captured within a blotting image. This study is dedicated to demonstrate that electrophoresis can be interpreted to execute a variant of *bucket sort* which ascendingly arranges the involved charged molecules by their masses. In concert with the notion of a blotting membrane system, we exploit the aforederived function $s(m, t)$ for geometrical definition of the buckets, each of them representing a region on the gel assigned to a cluster. For the experiment, we utilise a predefined mixture of double-stranded DNA whose strand lengths reach from 100 base pairs (bp) up to 1000bp in steps of 100bp. Additionally, the sample contains strands with 1200bp and 1500bp. The whole sample acts as a so-called DNA ladder made from a cleaved plasmid (100bp standard

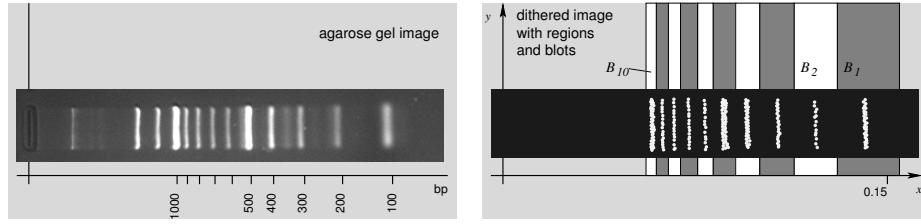


Fig. 7. Left: 100bp ladder with DNA bands visible in agarose gel. Bands at 500 and 1000bp emphasized by enhanced DNA concentration. **Right:** Dithered image taken from gel photo. Based on an underlying coordinate system in metre scale, altering background regions B_1 up to B_{10} define regions (buckets) in which the DNA molecules accumulate forming bands arranged as non-overlapping blots. DNA bands with more than 1000bp neglected in dithered image.

ladder by New England BioLabs) composed of an almost uniform distribution of the nucleotides A, C, G, and T.

In order to disclose the relation between mass of a DNA double strand and its length in base pairs, we need to consider the average mass of a nucleotide. Indeed, there are slight mass deviations between single nucleotides A (Adenine, $\approx 5.467 \cdot 10^{-25}$ kg), C (Cytosine, $\approx 5.234 \cdot 10^{-25}$ kg), G (Guanine, $\approx 5.732 \cdot 10^{-25}$ kg), and T (Thymine, $\approx 5.301 \cdot 10^{-25}$ kg). Each nucleotide mass comprises the chemical base together with its section of the sugar-phosphate backbone. In average, we obtain $\approx 5.4335 \cdot 10^{-25}$ kg per nucleotide or $\approx 1.0867 \cdot 10^{-24}$ kg per base pair. Marginal influences of dye and ethidium bromide are neglected.

The left part of Figure 7 depicts the agarose gel image under UV light making visible the individual DNA bands arranged within the lane after running the electrophoresis for 2700 seconds (45 minutes). Each band corresponds to a predefined strand length. Using the gel image, a simple *dithering* (selectively applied to all bands between 100 and 1000 base pairs) generates a set of uniquely located dots from the bands, see right part of Figure 7. The dots are equipped with geometrical coordinates. At that stage, we have everything at hand to finally formulate the blotting membrane system from the experiment (running conditions $E = 400 \frac{\text{V}}{\text{m}}$, $\eta = 0.001 \frac{\text{kg}}{\text{m}\cdot\text{s}}$):

$$\begin{aligned} \boxed{\Pi} &= (P, L, C, B_1, \dots, B_{|C|}, S, R, r) \quad \text{with} \\ L &= \{l\} \\ P &= \{(0.143, 0.082, l), (0.142, 0.077, l), \dots, (0.714, 0.079, l)\} \\ C &= \{1, \dots, 10\} \\ B_i &= \{(x, y, l) \mid (x, y, l) \in P \wedge \\ &\quad s((100 \cdot i - 50) \cdot 1.0867 \cdot 10^{-24}, 2700) \geq x \wedge \\ &\quad x \geq s((100 \cdot i + 50) \cdot 1.0867 \cdot 10^{-24}, 2700)\} \quad \forall i \in C \\ S(c) &= |B_c| \quad \forall c \in C \\ R &= \mathbb{N}^{|C|} \\ r(S) &= (S(1), \dots, S(10)) \end{aligned}$$

5 Discussion and Conclusions

Along with a couple of case studies we demonstrated the descriptive practicability of blotting membrane systems. The main advantage of this formalism consists in its capability of tremendous data reduction. From a large number of geometrical dot coordinates together with some auxiliary data, a rather condensed systems output can be achieved retaining the crucial behaviour and characteristics of the system under study. For instance, in case of numerical integration, an amount of several million individual dots taken as systems input becomes compiled forming a single rational number which stands for the resulting integral value.

In its present form, blotting membrane systems behave in a more or less *static* manner. Recently, they lack any temporal or dynamical aspect [1, 12, 17]. A promising extension could take into account a possible *progression* of the blots, spots, and dots over time. From a modelling point of view, this feature might be incorporated by creation of a finite automaton whose states are blotting membrane systems. Using periodical trigger signals or by means of signalling events, a dedicated state transition from the previous system to its successor(s) could help to trace clusters together with its counts along a time line.

We believe that membrane computing as an innovative field of research sustainably benefits from a plethora of real-world applications making the underlying algebraic formalisms a powerful toolbox to cope with challenges in managing big data. Categorised counting can be seen as a technique of data mining which combines the possibility of massively parallel data processing promoted in membrane computing with exploitation of statistical or deductive methods. Futural studies will be intent upon strengthening this fruitful relationship.

References

1. R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini, L. Tesei. Spatial P Systems. *Natural Computing* **10**:3-16, 2011
2. F. Bernardini, M. Gheorghe, N. Krasnogor. Quorum sensing P systems. *Theoretical Computer Science* **371(1-2)**:20-33, 2007
3. B. Bowen, J. Steinberg, U.K. Laemmli, H. Weintraub. The detection of DNA-binding proteins by protein blotting. *Nucleic Acids Research* **8(1)**:1-20, 1980
4. M. Chalfie, Y. Tu, G. Euskirchen, W.W. Ward, D.C. Prasher. Green fluorescent protein as a marker for gene expression. *Science* **263(5148)**:802-805, 1994
5. L.S. Churchman, Z. Ökten, R.S. Rock, J.F. Dawson, J.A. Spudich. Single molecule high-resolution colocalization of Cy3 and Cy5 attached to macromolecules measures intramolecular distances through time. *PNAS* **102(5)**:1419-1423, 2005
6. H. Cohen, C. Lefebvre (Eds.). *Handbook of Categorization in Cognitive Science*. Elsevier, 2005
7. T.E. Dawson, J.S. Pate. Seasonal water uptake and movement in root systems of plants of dimorphic root morphology: a stable isotope investigation. *Oecologia* **107**:13-20, 1996
8. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. From Data Mining to Knowledge Discovery in Databases. *American Assoc. for Artificial Intelligence* **3**:37-54, 1996

9. D. Hames, N. Hooper. *Biochemistry*. Third Edition, Taylor & Francis, 2005
10. J. Han, M. Kamber, J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011
11. T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Series in Statistics, Springer Verlag, 2009
12. T. Hinze, R. Fassler, T. Lenser, P. Dittrich. Register Machine Computations on Binary Numbers by Oscillating and Catalytic Chemical Reactions Modelled using Mass-Action Kinetics. *International Journal of Foundations of Computer Science* **20(3)**:411-426, 2009
13. T. Hinze, B. Schell, M. Schumann, C. Bodenstein. Maintenance of Chronobiological Information by P System Mediated Assembly of Control Units for Oscillatory Waveforms and Frequency. In E. Cshaj-Varju et al. (Eds.). Proc. CMC13. *Lecture Notes in Computer Science* **7762**:208-227, 2013
14. T. Hinze, J. Behre, C. Bodenstein, G. Escuela, G. Grünert, P. Hofstedt, P. Sauer, S. Hayat, P. Dittrich. Membrane Systems and Tools Combining Dynamical Structures with Reaction Kinetics for Applications in Chronobiology. In P. Frisco, M. Gheorghe, M.J. Perez-Jimenez (Eds.). *Applications of Membrane Computing in Systems and Synthetic Biology. Series Emergence, Complexity, and Computation*. Vol. 7, pp. 133-173, Springer Verlag, 2014
15. B.G. Johansson. Agarose Gel Electrophoresis. *Scandinavian Journal of Clinical and Laboratory Investigation* **29(s124)**:7-19, 1972
16. G.J. Kremers, S.G. Gilbert, P.J. Cranfill, M.W. Davidson, D.W. Piston. Fluorescent proteins at a glance. *Journal of Cell Science* **124**:157-160, 2011
17. L. Marchetti, V. Manca, R. Pagliarini, A. Bollig-Fischer. MP Modelling for Systems Biology: Two Case Studies. In P. Frisco, M. Gheorghe, M.J. Perez-Jimenez (Eds.). *Applications of Membrane Computing in Systems and Synthetic Biology. Series Emergence, Complexity, and Computation*. Vol. 7, pp. 223-243, Springer Verlag, 2014
18. C.L. Marlatt. The periodical cicada. *Bull. U.S. Dept. Agri., Div. Entomol. Bull.* **18**:52, 1907
19. M.B. Miller, B.L. Bassler. Quorum Sensing in Bacteria. *Annu. Rev. Microbiol.* **55**:165-199, 2001
20. G. Păun. *Membrane Computing: An Introduction*. Springer Verlag, 2002
21. R.W. Sabnis. *Handbook of biological dyes and stains: synthesis and industrial application*. Wiley-VCH, 2010
22. E.M. Southern. Detection of specific sequences among DNA fragments separated by gel electrophoresis. *Journal of Molecular Biology* **98(3)**:503-517, 1975
23. A. Sparmann, M. van Lohuizen. Polycomb silencers control cell fate, development and cancer. *Nature Reviews Cancer* **6**:846-856, 2006
24. H. Towbin, T. Staehelin, J. Gordon. Electrophoretic transfer of proteins from polyacrylamide gels to nitrocellulose sheets: Procedure and some applications. *PNAS* **76(9)**:4350-4354, 1979
25. C. Nüsslein-Volhard. Determination of the embryonic axes of Drosophila. *Development* **113**:1-10, 1991
26. M. Lizotte-Waniewski, W. Tawe, D.B. Guiliano, W. Lu, J. Liu, S.A. Williams, S. Lustigman. Identification of Potential Vaccine and Drug Target Candidates by Expressed Sequence Tag Analysis and Immunoscreening of *Onchocerca volvulus* Larval cDNA Libraries. *Infection and Immunity* **68(6)**:3491-3501, 2000
27. R. Westermeier. *Electrophoresis in Practice*. Wiley-VCH, 2005
28. K.S. Williams, C. Simon. The ecology, behavior and evolution of periodical cicadas. *Annual Review of Entomology* **40**:269-295, 1995

Polymorphic P Systems with Non-cooperative Rules and No Ingredients

Sergiu Ivanov

Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
61, av. du gén. de Gaulle, 94010 Créteil, France
`sergiu.ivanov@u-pec.fr`

Abstract. Polymorphic P systems represent a variant of the bio-inspired computational model of P systems, in which the rules are not explicitly given in the description of the system, but are implicitly defined by the contents of certain membranes. In this paper we give a characterisation of the most basic class of such systems, in which only non-cooperative rules are allowed and no ingredients are included. We start by formulating two different formal definitions of non-cooperativity and then show that they are equivalent. We also define the upper bound on the generative power of polymorphic P systems and, finally, show that the languages produced by such systems form a hierarchy related to the maximal allowed depth of the membrane structure.

1 Introduction

Membrane computing is a fast-growing research field opened by Gh. Păun in 1998. It presents a formal framework inspired from the structure and functioning of the living cells. In the paper [1], yet another relatively powerful extension to the model is defined, which allows the system to dynamically change the set of rules, which is thus not limited to some finite prescribed set of candidates. There were three main motives for this extension. Firstly, experience shows that “practical” problems need “more” computing potential than just computational completeness. Secondly, a very important computational ingredient was imported from computer science: the approach in which both the “program” and the “data” are represented in the same way. And finally, such an extension correlates with the biological idea that different actions are carried out by different objects which can, too, in their turn, be acted upon. The full motivation as well as references to related papers are given in the cited work [1].

In this paper we give a characterisation of the most basic class of polymorphic P systems, which only relies on non-cooperative rules and does not include any ingredients. We start by formulating two different formal definitions of non-cooperativity and then show that they are equivalent. We also define the upper bound on the generative power of polymorphic P systems and, finally, show that the languages produced by such systems form a hierarchy related to the maximal allowed depth of the membrane structure.

The motivation for examining the class of polymorphic P systems with non-cooperative rules without any additional ingredients but polymorphism itself is twofold. On the one hand, since this is the most restricted variant of polymorphic P systems, and thus understanding its computing power is important for understanding the computing power of the general variant. On the other hand, polymorphic P systems without ingredients and with non-cooperative rules turn out to be more powerful than conventional transition P systems with non-cooperative rules. In fact, polymorphism enables some otherwise very restricted models to generate rather “difficult” superexponential number languages. Given that an actual software implementation of polymorphic rules does not seem to require essentially more resources than that of invariant rules, restricted variants of polymorphic P systems may turn out practical to use in solving certain real-world problems. We would expect to find such applications in the domains in which massive parallelism is required, because non-cooperative rules are intrinsically easy to parallelise.

2 Preliminaries

2.1 Formal Languages and Complexity Theory

In this section we recall some of the basic notions of the formal language and complexity theories. For a more comprehensive overview of the mentioned topics we refer the reader to [3, 7, 8, 2].

A (non-deterministic) *finite automaton* is the tuple

$$A = (Q, \Sigma, \delta, q_0, F),$$

where Q is a finite set of states, Σ is a finite set of input symbols, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, q_0 is the initial state of the automaton and F is the set of final states. The automaton starts in the initial state q_0 and examines the input tape symbol by symbol. If it finds the symbol $a \in \Sigma$ in state $q_i \in Q$, it transitions in a non-deterministic way into one of the states from the set $\delta(q_i, a)$. If, at a certain step, the automaton is in state q_i and is reading a symbol a such that $\delta(q_i, a) = \emptyset$, it halts. The automaton A is said to *accept* (recognise) only those inputs for which it consumes the input entirely and halts in a final state.

We will refer to the families of sets which can be recognised by a finite automaton as *regular families*.

Let V be a finite set. A *finite multiset* w over V is a mapping $w : V \rightarrow \mathbb{N}$, which specifies the number of occurrences of each $a \in V$. The size of the multiset is defined as $|w| = \sum_{a \in V} w(a)$. A multiset w over V can be also represented by any

string x such that it contains exactly $w(a)$ symbols a , for all $a \in V$. The *support* of w is the set of symbols which appear in it: $\text{supp}(w) = \{a \in V \mid w(a) > 0\}$. Given two multisets x and y over the same alphabet V , x is called a submultiset of y , written $x \leq y$, if $x(a) \leq y(a)$, for all $a \in V$. If $x(a) < y(a)$ for some $a \in V$, x is called a strict submultiset of y , written as $x < y$.

Let f and g be two functions defined on a subset of real numbers. We write $f(x) \in O(g(x))$ if and only if there exists a positive real number M and a real number x_0 such that $|f(x)| \leq M|g(x)|$ for all $x \geq x_0$. We write $f(x) \in \Theta(g(x))$ if there exist positive real numbers k_1 and k_2 , and a real number x_0 , such that $k_1 \cdot g(x) \leq f(x) \leq k_2 \cdot g(x)$, for all $x \geq x_0$.

We will now define the notion of a polymorphic P system without ingredients. For the original definition we refer the reader to [1]. For a general introduction to P systems, the reader is referred to [4, 6]. For a comprehensive overview of the domain, we recommend the handbook [7].

A *polymorphic P system* is defined as a tuple

$$\Pi = (O, T, \mu, w_s, w_{1L}, w_{1R}, \dots, w_{mL}, w_{mR}, i_{out}),$$

where O is a finite alphabet and μ is a tree structure consisting of $2m + 1$ membranes, bijectively labelled by the elements of $H = \{s\} \cup \{iL, iR \mid 1 \leq i \leq m\}$. The label s is assigned to the skin membrane. We require that, for $1 \leq i \leq m$, the membranes iL and iR have the same parent membrane. Finally, the set $T \subseteq O$ describes the output objects, while $i_{out} \in H \cup \{0\}$ gives the output region, the symbol 0 standing for the environment.

The rules of a polymorphic P system are not explicitly given in its description. Essentially, such a system has m rules, and these rules change as the contents of the regions other than the skin change. Initially, for $1 \leq i \leq m$, the rule $i : w_{iL} \rightarrow w_{iR}$ belongs to the region defined by the parent membrane of iL and iR . If w_{iL} is empty, then the rule is considered *disabled*. For every step of the computation, each rule is defined in the same way, taking the current contents of iL and iR instead of the initial ones. We sometimes refer to the membranes iL and iR as to left-hand-side and right-hand-side membranes of the rule i , respectively.

For reasons of readability, we will often resort to *graphical presentation* of polymorphic P systems. In such figures, we will not draw the membranes corresponding to invariable rules, but instead will write the rules directly, as it is conventionally done for other P system models.

A polymorphic P system Π of degree n is said to be *with strongly non-cooperative rules*, if, in any evolution, any of the membranes iL , $1 \leq i \leq m$, contains at most one symbol. A polymorphic P system Π of degree n is said to be *with weakly non-cooperative rules*, if, in any evolution, all rules which are applied have exactly one symbol in the left-hand side.

Note that weak non-cooperativity allows left-hand sides of rules to contain more than one symbol and only requires that, whenever this happens, the rule be not applicable.

In our shorthand notation for classes of polymorphic P systems we will write $ncoo_s$ and $ncoo_w$ to refer to the classes of polymorphic P systems with strongly and weakly non-cooperative rules respectively. We will also specify the number of membranes and whether disabling rules is allowed. Thus to refer to the family of polymorphic P systems with weakly non-cooperative rules, in which rule disabling is allowed, and which have at most k membranes, we will write the

following expression:

$$OP_k(\text{polym}_{+d}(\text{ncoo}_w)).$$

If no bound on the number of membranes is specified, k is replaced by $*$ or is omitted. If disabling rules is not allowed, we write $-d$ in the subscript of *polym*.

For weakly non-cooperative polymorphic P systems, *not* allowing the disabling of rules would mean that all left-hand-side membranes are *not* empty at any step of the evolution. In the case of strong non-cooperativity, the same requirement would mean that no left-hand-side membrane iL contains an erasing rule (a rule of the form $a \rightarrow \lambda$).

We would like to remark that, due to the inherently dynamic nature of rules of a polymorphic P system, verifying whether the rules are weakly non-cooperative, or whether they can be disabled, are not straightforward tasks. Indeed, deciding if the rules are never disabled would mean checking if the languages generated in left-hand-side membranes contain the empty word. Proving weak non-cooperativity is even more complex: it would require showing that, whenever the contents of the left-hand-side membrane iL contain more than one symbol, this multiset is *not* a submultiset of the contents of the parent membrane of iL .

The *depth* of a polymorphic P system is defined as the height of the membrane structure μ seen as a tree. Thus, a polymorphic P system which has no rules has depth 1, a polymorphic P systems which has invariable rules is of depth 2, etc. To refer to a class of polymorphic P system of depth limited to d , we add the number d as a superscript the notation we have introduced above:

$$OP_k^d(\text{polym}_{+d}(\text{ncoo}_w)).$$

A polymorphic P system is called *left-polymorphic* if only left-hand sides of the rules are allowed to vary. A polymorphic P system is called *right-polymorphic* if only right-hand sides of the rules are allowed to vary.

In the notation we have introduced in the previous paragraphs, we will use the symbols *lpolym* and *rpolym* to refer to the classes of left- and right-polymorphic P systems respectively.

Note that the effective implication of the definition of left-polymorphic P systems is that no right-hand-side is allowed to contain any rules. The symmetric statement is true for right-polymorphic P systems.

3 Strong and Weak Non-cooperativity

In this section we are going to show that the notions of strong and weak non-cooperativity are equivalent, when no ingredients but disabling of rules are allowed.

We start by proving an intermediate result concerning left-polymorphic P systems of depth 3.

Lemma 1. *Consider a left-polymorphic P system Π of depth 3 and with weakly non-cooperative rules. Then, any weakly non-cooperative rule i of Π can be replaced with a strongly non-cooperative one without changing the generated language.*

Proof. Consider a left-hand side membrane iL of Π . If the contents of this membrane do not change in any evolution, or do not ever include more than one symbol, the rule i is strongly non-cooperative. Suppose now that, at certain evolution steps iL may contain more than one symbol. Consider a computation \mathcal{C} of Π and two configurations C_k and C_m , $k < m$, such that in C_k and C_m the membrane iL contains at most one symbol, and none of the configurations C_{k+1}, \dots, C_{m-1} has this property. Suppose that in C_k the membrane iL contains an instance of x and in C_m it contains an instance of y or the empty multiset λ . Remember that, since the depth of Π is 3, the rules which appear in iL are invariable, that is, they are normal context-free multiset rewriting rules. The fact that such rules cannot be used for counting multiplicities of symbols motivates the following construction.

Define the transition relation $(\rightarrow_{iL}) \subseteq 2^O \times 2^O$ on the subsets of the alphabet O of Π as follows. Consider the “flattened” version of the rules in iL , that is, only consider the supports of the multisets defining the right-hand sides of the rules. Now, for $M_1, M_2 \subseteq Q$, define $M_1 \rightarrow_{iL} M_2$ to be equivalent to the fact that the set M_2 can be obtained from M_1 as a union of the right-hand sides of those (flattened) rules in iL whose left-hand sides are in M_1 .

Consider a sequence $(M_i)_{1 \leq i \leq k}$ of subsets of O , such that $M_i \rightarrow_{iL} M_{i+1}$, $1 \leq i \leq k-1$. Then, whenever iL contains a multiset with support M_1 there exists an evolution of Π in which, after k steps, iL contains a multiset with support M_2 . Such an evolution can be obtained by always applying one kind of rule to any symbol; e.g., if there are several instances of a and several rules with a in their left-hand side, then we only apply one of the rules to all instances of the symbol a .

Now turn back to our original pair of configurations C_k and C_m , and consider the case in which, in C_m , iL contains the empty multiset. Some of the possible evolutions of the contents of iL are given by the sequences \mathcal{M} of subsets of O which pairwise satisfy the relation (\rightarrow_{iL}) and in which the first subset is $\{x\}$ and the last one is \emptyset . We now claim that the *length* of a derivation of Π which brings C_k into C_m corresponds to the length of such a sequence \mathcal{M} . Indeed, the evolutions of the contents of iL which are *not* directly modelled by chains defined by (\rightarrow_{iL}) are those in which, at a certain step, different rules r_1, \dots, r_m are applied to different instances of the same symbol a . But then, the next state of iL can be seen as modelled by *all* subsets of O which correspond to the (possible) steps in which only one of the rules r_1, \dots, r_m is applied.

Consider, for example, the multiset aa and two rules $a \rightarrow b$ and $a \rightarrow c$. This gives rise to the following two relations: $\{a\} \rightarrow \{b\}$ and $\{a\} \rightarrow \{c\}$. Then we say that these two relations model any transition from aa to a multiset derived according to the shown rules.

Using this approach we can model *any* evolution of iL as a *tree* of subsets of O . The height of this tree will be given by the length of the longest path from its root, which is itself a sequence of subsets of O pairwise satisfying (\rightarrow_{iL}) . Thus, the length of any derivation bringing an instance of x in iL to an empty set is given by a length of a such sequence of subsets. Remember now that (\rightarrow_{iL}) is a

finite relation, which means that there exists a finite automaton which recognises the sequences defined by (\rightarrow_{iL}) .

To prove that a similar fact holds for the case in which the instance of x in iL is transformed into exactly one instance of y in configuration C_m we will define the extended transition relation $(\rightarrow'_{iL}) \subseteq 2^{O'} \times 2^{O'}$, where $O' = \{a, a' \mid a \in O\}$. Intuitively, the fact that a set $M \subseteq O'$ contains a primed symbol a' will refer to the fact that M represents a multiset that contains more than one instance of a . The extended transition relation is defined in the following way.

For $M \subseteq O$, we consider all the multisets w which can be obtained by applying the rules from iL to the multiset containing one instance of every symbol from M . Then, for every such multiset w we build the set $M' \subseteq O'$ in the following way:

$$M' = \{a \in O \mid w(a) = 1\} \cup \{a' \in O' \mid w(a) > 1\},$$

and we put $M \rightarrow'_{iL} M'$ for every such M' .

Now let $M \subseteq O'$ contain some primed symbols. First off, let R be the set of “flattened” rules in iL (only take the supports of their right-hand sides); consider also the set R' which contains the same rules as in R , but operating on *primed* symbols. We now build all the multisets which can be obtained by applying the rules from $R \cup R'$ to the multiset containing one instance of every symbol from M . For every such multiset w we build the set M' in the following way:

$$M' = \{a \in O \mid w(a) = 1\} \cup \{a' \in O' \mid w(a) > 1\} \cup \{a' \in O' \mid w(a') > 0\},$$

that is, when we apply a rule to a primed symbol, all the products are kept primed to indicate that, in the *represented* multiset there may be multiple instances of them. Again, we put $M \rightarrow'_{iL} M'$ for all such sets M' .

The extended transition relation as we have defined it so far (\rightarrow'_{iL}) allows us to model the possible evolution scenarios of the contents of the region iL in a way similar to (\rightarrow_{iL}) . However, it does not yet capture the possibility of deriving a *single* instance of y from a different symbol a present in more than one copy in iL . To capture that possibility as well, we will further extend (\rightarrow'_{iL}) . Based on the original set M and a primed symbol $a' \in M$, we will define another set $M_a = (M \cup \{a\}) \setminus \{a'\}$ and we will construct the sets M'_a in exactly the same way we constructed M' . We will then pair up those sets M' and M'_a which were obtained by applying two different rules (modulo primed symbols) to a' (respectively a), build the unions $M' \cup M'_a$ of such pairs, and set $M \rightarrow'_{iL} M' \cup M'_a$. The way in which these unions are constructed permits us to capture the situations in which only one instance of a is consumed by a rule, while the other a 's were consumed by different rules.

Repeating the argument we made above about (\rightarrow_{iL}) for the situation in which C_m contains the empty multiset, we can conclude that the number of steps between the configurations in which iL contains one instance of x and those in which it contains exactly one y correspond to the length of chains defined by (\rightarrow'_{iL}) starting at $\{x\}$ and ending at $\{y\}$. Therefore, there exists a finite automaton recognising the heights of such trees and only them.

From the existence of the finite automata recognising exactly the distances in steps between the configurations in which iL contains one symbol or the empty set, it follows that it is possible to replace the rules in iL by invariable rules which have at most one symbol in their right-hand side and which model the state transitions of the corresponding finite automaton, which implies the statement of the lemma.

The previous lemma makes it possible to prove the following statement concerning left-polymorphic P systems.

Theorem 1. $NOP_*(lpoly_{+d}(ncoo_w)) = NOP_*(lpoly_{+d}(ncoo_s))$.

Proof. Consider a left-polymorphic P system Π with weakly non-cooperative rules. We will prove the statement of the theorem by induction over the left-hand-side membranes of Π . Our goal will be to describe the construction of the counterpart of the extended transition relation from Lemma 1.

Consider such a membrane iL of Π that the sequences of symbols which may appear in the left-hand sides of the rules in iL at the moments when they are applicable can be recognised by a finite automaton (one per rule). Remember that the right-hand sides of all the rules in Π are constant. On the other hand, the fact that the rules in iL are strongly non-cooperative implies that the total number of rules which may appear in iL is *finite*. This makes it possible to construct finite (multi)sets of rules available in iL at a step of evolution. Furthermore, all the sequences of rules made available by a pair of membranes jL and jR contained inside iL are regular, and so are the sequences of (multi)sets of rules made available in iL .

We will construct the transition relation starting from the first step of evolution of iL . We will represent the contents at this step as a subset M of the alphabet O' extended with prime symbols and will list the sets M' as we did in the proof of Lemma 1, using the rules available at this step of evolution. Since the sequences of sets of available rules are regular and since O' is a finite set, the sequences of sets defined by the newly-built transition relation are regular as well.

We can now conclude that, if all the sequences of symbols which appear in the left-hand sides of the rules in iL are regular, the distances between the steps in which iL contains exactly one symbol or no symbols at all are regular. Therefore it is possible to replace the rules in iL with invariable rules modelling the transitions of the said finite automaton. Together with Lemma 1, this observation implies the statement of the theorem.

We would now like to prove the general result that polymorphic P systems with *weakly* non-cooperative rules do not generate more number languages than polymorphic P systems with *strongly* non-cooperative rules. It suffices to show that the sequence of supports of the multisets representing the right-hand side of a rule is regular. The following lemma captures this assertion.

Lemma 2. *Consider a weakly non-cooperative polymorphic P system Π , pick a right-hand-side membrane iR of Π , and consider the sequence of multisets*

$(w_i)_{1 \leq i \leq n}$ that appear in iR during an n -step derivation. The sequence of supports of these multisets is regular.

Proof. In the proof of Lemma 1 we have already seen that the statement we would like to prove is valid for a membrane iR which only contains invariable rules. Consider now such a membrane iR which contains rules whose right-hand sides satisfy the statement of the present lemma and whose left-hand sides can be modelled with one-symbol transitions (as in Theorem 1). We can adapt the procedure we followed in the proof of the previous theorem to the situation in the present lemma. Indeed, since the sequences of supports of both left-hand sides and right-hand sides of the rules in iR are regular, it is possible to build a regular sequence of (multi)sets of rules available in iR at every step of the evolution. Then a sequence of supports of the multisets which appear in iR will be defined by its first element and the sequence of rules available at each step, and will therefore be regular. Inductively applying this observation in the bottom-up manner to all right-hand side membranes of Π proves the statement of the lemma.

All the arguments we have shown by now can be used to directly derive the following statement showing that strongly and weakly non-cooperative polymorphic P systems define the same sets of languages.

Theorem 2. $NOP_*(polym_{+d}(ncoo_w)) = NOP_*(polym_{+d}(ncoo_s))$.

Proof. Similarly to the procedure shown in Theorem 1, the proof here is done by induction over left-hand-side membranes. The idea is essentially the same, and is also based on the statement of Lemma 2: supposing that the sequences of (multi)sets of flattened rules appearing in a given left-hand-side membrane are regular, it follows that the activity of this membrane can be simulated with a finite automaton, which proves the statement of the theorem.

Given the fundamental similarity between P systems with strongly and weakly non-cooperative rules, we will from now on refer to both classes using the word “non-cooperative” and the notation $ncoo$ without a subscript.

The proofs we have shown so far imply the following important corollaries.

Corollary 1. *Given a polymorphic P system $\Pi \in OP_*(polym_{+d}(ncoo))$ and a membrane i of it, the sequence of supports of the multisets which appear in this membrane is regular.*

Corollary 2. *Given a polymorphic P system $\Pi \in OP_*(polym_{+d}(ncoo))$, it is possible to construct another polymorphic P system Π' such that $N(\Pi') = N(\Pi)$ and all left-hand-side membranes of Π' contain invariable rules.*

The P system Π' from the previous corollary, is given by the construction briefly presented in the proof of Theorem 2, that is, by replacing all rules in a left-hand-side membrane with invariable rules simulating the transitions of a finite automaton. The practical implication of this corollary is that, in the case of

polymorphic P systems with non-cooperative rules and no additional ingredients, the variability of left-hand sides of rules is only useful for slightly changing the scope of a rule and switching rules on and off.

We conclude this section by showing that, in the case of non-cooperative rules, explicitly disabling a rule by emptying its left-hand-side membrane can be simulated by replacing the empty multiset with a special symbol not belonging to the original alphabet.

Lemma 3. $NOP_*(polym_{-d}(ncoo)) = NOP_*(polym_{+d}(ncoo))$.

Proof. Take a polymorphic P system $\Pi \in OP_*(polym_{+d}(ncoo))$ and pick a left-hand-side membrane iL which becomes empty at a certain step of the computation. According to Corollary 2, we can consider that all the rules in iL are invariable, without losing generality. Now replace all the empty right-hand sides of rules in iL with the singleton multiset \perp , where \perp does not belong to the alphabet of Π . Clearly, whenever \perp appears in iL , the rule i is effectively disabled and is blocked in this state forever. On the other hand, we avoid emptying the membrane iL , which proves the statement of the theorem.

Because it turns out that disabling of rules makes no real difference in terms of the number languages a (left-) polymorphic P system with non-cooperative rules can generate, we will sometimes avoid adding “+ d ” (“- d ”) to the shorthand notation for classes of polymorphic P systems with non-cooperative rules.

4 Left Polymorphism

In this section we will briefly overview the computational power of left-polymorphic P systems. Remember that, in such P systems, the right-hand side of any rule is not allowed to embed any other rules. We start by showing that such polymorphic P systems are still more powerful than conventional transition P systems [7].

Lemma 4. $L_{2^n} = \{2^n \mid n \in \mathbb{N}\} \in NOP_*(lpolym(ncoo))$.

Proof. We construct the following left-polymorphic P system generating the number language L_{2^n} :

$$\begin{aligned} \Pi &= (\{a\}, \{a\}, \mu, a, aa, a, a, a, \lambda, s), \text{ where} \\ \mu &= [[[]_{2L} []_{2R} []_{3L} []_{3R}]_{1L} []_{1R}]_s. \end{aligned}$$

The graphical representation of this P system is given in Figure 1.

Π works by repeatedly doubling the number of a 's in the skin membrane, until the symbol a in $1L$ is rewritten into the empty multiset, which disables rule 1 and makes Π halt.

On the other hand, the following observation shows that left-polymorphic P systems cannot generate all recursively enumerable sets of numbers.

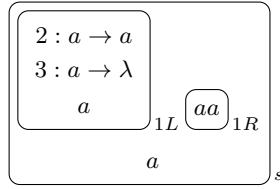


Fig. 1. A left-polymorphic P system generating $\{2^n \mid n \in \mathbb{N}\}$

Lemma 5. $L_{n!} = \{n! \mid n \in \mathbb{N}\} \notin \text{NOP}_*(\text{lpoly}_{+d}(\text{ncoo}))$.

Proof. We will pick a left-polymorphic P system $\Pi \in \text{NOP}_*(\text{lpoly}(\text{ncoo}))$ and see what are the possible multiplicities of symbols in some of the multisets it generates. There exist derivations in which no more than one type of rule is applied per symbol type (e.g., only the rule $a \rightarrow bc$ is applied 5 times to all the 5 instances of a , even though there may be more rules consuming the same symbol). For such derivations, the multiplicities of all symbols in the halting configuration can be expressed as sums of products of the quantities of certain symbols in the initial contents of the skin by constant factors, which depend only on the (invariable) right-hand sides of the rules. Since generating the factorial would require multiplication by an *unbounded* set of factors, the preceding observation concludes the proof.

Note that Lemma 5 only shows an example of a language which cannot be generated by left-polymorphic P systems; this model of P systems can still be undecidable.

5 Right Polymorphism

In this section we will show that right-polymorphic P systems, just as their left-polymorphic counterparts, generate a wider class of languages than conventional transition P systems.

Lemma 6. $L'_{2^n} = \{2^n \mid n \in \mathbb{N}, n > 2\} \in \text{NOP}_*(\text{rpolym}(\text{ncoo}))$.

Proof. We will construct the following right-polymorphic P system generating the number language L'_{2^n} :

$$\begin{aligned} \Pi &= (\{a, b\}, \{b\}, \mu, a, a, aa, a, a, a, a, b, s), \text{ where} \\ \mu &= [[]_{1L} [[]_{2L} [[]_{3L} [[]_{3R} [[]_{4L} [[]_{4R}]_{2R}]_{1R}]_s. \end{aligned}$$

The graphical representation of this P system is given in Figure 2.

The principle behind the functioning of this polymorphic P system is essentially the same as the one used in the proof of Lemma 4: Π works by repeatedly doubling the number of a 's in the skin. Having Π halt correctly is trickier, though, since we cannot disable rules and we cannot just rewrite the symbols in

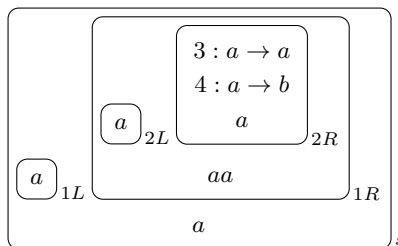


Fig. 2. A right-polymorphic P system generating $\{2^n \mid n \in \mathbb{N}, n > 2\}$

$1R$ to b , because then we could get rule 1 to have the form $a \rightarrow ab$, which would result in the production of the language $\{m \cdot 2^n \mid m, n \in \mathbb{N}\}$ (maybe without several shortest words) instead of L'_{2^n} . To work around this problem we allow *no choice* in membrane $1R$, that is, all instances of a are *always* rewritten into something, and it is the right-hand side of rule 2 that decides whether to reproduce the two instances of a and keep the exponentiation going, or transform them into b and stop the evolution. During the time b 's take to propagate through the membrane structure, Π keeps multiplying the number in the skin by 2, which results in the fact that Π cannot generate the numbers 2 and 2^2 from $L_{2^n} \setminus L'_{2^n}$.

The upper bound on the generating power of right-polymorphic P systems is obtained in the next section by establishing an upper bound on the expressive power of general non-cooperative polymorphic P systems.

6 General Polymorphism

In this section we will show that there are languages which cannot be generated by polymorphic P systems with non-cooperative rules. The intuition for this observation comes from the fact that non-cooperative rules cannot *synchronise* separate processes running in the P system. Therefore, it suffices to pick a language which cannot be generated without such synchronisation. The following theorem formally captures this intuition.

Theorem 3. $L_{n!} = \{n! \mid n \in \mathbb{N}\} \notin \text{NOP}_*(\text{polym}(n\text{coo}))$.

Proof. Suppose there exists a polymorphic P system $\Pi \in \text{OP}_*(\text{polym}(n\text{coo}))$ which generates the factorial language. In this case, according to Lemma 5, the rules of Π must rely on variable right-hand sides, and there must exist such rules which are applied an unbounded number of times.

Consider such a rule i in skin and its right-hand-side membrane iR which contains the multiset w_k at step k . Let $p_k < w_k$ be the bounded multiset of symbols which may be consumed by rules with right-hand sides longer than 1. Remember now that, at a certain moment, the membrane iR must *halt*. Therefore, for all symbols in p_k , it is possible to choose such rules which will lead to halting of iR in a bounded number of steps.

Suppose now that the symbols in $w_k - p_k$ are not consumed by any rules in iR . This means that the symbols from w_k may be consumed by some rules in the skin, with indices from the (not necessarily non-empty) set I (including, maybe, the rule i). Since, according to our supposition, the rule i is applied an unbounded number of times, the rules from I must be capable of consuming the symbols from w_k at an unbounded number of computation steps. But then, if, at a certain step, the membrane iR halts, the rules from I and the left-hand side membrane iL may continue their operation, which will lead to an unbounded number of multiplications of the quantities of certain symbols in the skin by a *constant* factor, which means that, in this case $N(\Pi) \setminus L_{n!} \neq \emptyset$.

Now consider the situation in which the symbols from $w_k - p_k$ are consumed by some rules in iR . The important difference which may arise in this case as compared to the previous one is the scenario in which, at a certain moment, the multiset $w_k - p_k$ is rewritten into a different multiset that, when in the skin, will only evolve for a bounded number of steps (i.e., no more rules will be applicable to the symbols derived from this multiset after a constant number of steps). However, in this case the same synchronisation problem persists: if the symbols from p_k are rewritten in such a way that the contents of iR will stop growing after a bounded number of steps, and the symbols from $w_k - p_k$ are still being rewritten into something which may be further used in the skin to apply iR some more times, the size of the contents of the skin will be essentially multiplied by a constant factor an unbounded number of times, which leads us to the conclusion that Π cannot produce the factorial language.

The following statement concerning the computational power of *right-polymorphic* P systems is a direct consequence of the previous theorem.

Corollary 3. $L_{n!} = \{n! \mid n \in \mathbb{N}\} \notin \text{NOP}_*(\text{rpolym}(\text{ncoo}))$.

7 A Hierarchy of Polymorphic P Systems with Non-cooperative Rules

In this section we will show that the generative power of a polymorphic P system Π is essentially limited by its depth. The intuition for this remark comes from looking at the superexponentially growing P systems shown in [1] and from the observation that more nested rules means faster growth.

We briefly recall how superexponential growth can be achieved with polymorphic P systems.

Example 1. (cf. [1]) Consider the following (left-)polymorphic P system:

$$\begin{aligned} \Pi &= (\{a\}, \{a\}, \mu, a, a, a, a, aa, s), \text{ where} \\ \mu &= [[]_{1L} [[]_{2L} [[]_{2R}]_{1R}]_s. \end{aligned}$$

The graphical presentation of Π is given in Figure 3.

After one step of evolution, the skin of Π will still contain the multiset a , and $1R$ will contain a^2 . At the next step the skin will contain a^4 and $1R$ will

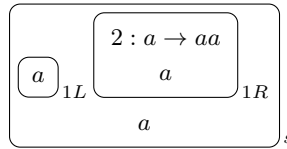


Fig. 3. A (left-)polymorphic P system with superexponential growth

contain a^4 . It is easy to verify that, after k evolution steps, the skin of Π will contain $2^{\frac{k(k-1)}{2}}$ instances of a .

While the polymorphic P system from the previous paragraph does have superexponential growth, it never halts. Actually *generating* a superexponential number language requires somewhat more design effort. A possible approach using the trick we have shown in the proof of Lemma 6 is implemented in the following example.

Example 2. The following P system generates the superexponential number language $\{2^{\frac{n(n-1)}{2}} \mid n \in \mathbb{N}, n > 3\}$.

$$\begin{aligned} \Pi &= (\{a, b\}, \{b\}, \mu, a, a, a, a, aa, a, a, a, a, b, s), \text{ where} \\ \mu &= [[]_{1L} [[]_{2L} [[]_{3L} [[]_{4L} [[]_{4R} [[]_{5L} [[]_{5R}]_{3R}]_{2R}]_{1R}]_s. \end{aligned}$$

The graphical representation of this P system is given in Figure 4.

Rules 1 and 2 of Π work exactly as the two rules of the P system shown in the previous example, while the other three rules are used to assure proper halting in the same way as it is done in the construction from Lemma 6.

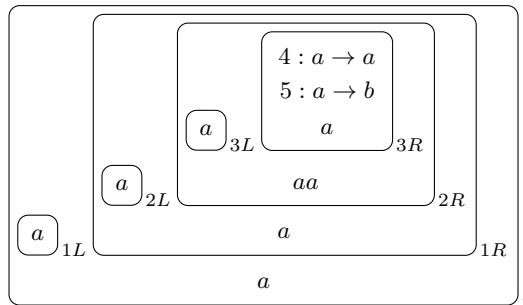


Fig. 4. A polymorphic P system generating $\{2^{\frac{n(n-1)}{2}} \mid n \in \mathbb{N}, n > 3\}$

We will now confirm the supposition of existence of an infinite hierarchy of polymorphic P systems by proving the following theorem.

Theorem 4. $L_{d-1} = \{2^{(n^{d-1})} \mid n \in \mathbb{N}\} \notin NOP_*^d(\text{polym}(n\text{coo}))$, for $d > 1$.

Proof. We will prove the statement by induction over the depth of polymorphic P systems. The base case $d = 2$ refers to transition P system with invariable rules, which as is known [7], cannot generate the language $L_1 = \{2^n \mid n \in \mathbb{N}\}$.

Before we proceed, we make an important observation as to how P systems with a fixed number of non-cooperative rules generate numbers. Consider two multisets a^{n_1} and a^{n_2} , where n_1 and n_2 are two different natural numbers greater than the number of rules in membrane i , and consider the possible ways they can evolve at the same step in this membrane. Because the left-hand sides of the rules are not allowed to contain more than one symbol, for any multiset w_1 derived from a^{n_1} there exists a multiset w_2 derived from a^{n_2} for which it is true that $\text{supp}(w_1) = \text{supp}(w_2)$ and $w_2(x) = (n_2 - n_1)w_1(x)$, for any $x \in w_1$. This essentially means that non-cooperative (polymorphic) rules cannot be used to *count* the number of instances of a symbol in a membrane. The further conclusion is that the sets of numbers that Π can generate are essentially given by the form of the right-hand sides of the rules.

Suppose now that the statement of the theorem is true for a certain d . Consider a polymorphic P system Π of depth $d + 1$. We pick a membrane iR of Π which, if regarded as a separate P system, has depth d . As it can be seen in Example 1, the fastest growth we can achieve for the contents of iR is having the *lengths* of the words at a step $f(n)$ of evolution belong to $O\left(2^{(n^{d-2})}\right)$, where $f(n) \in \Theta(n)$. But then we can use the rule i to assure that the growth of the contents of the skin as a function of the number of the evolution step is only in $O\left(2^{(n^{d-1})}\right)$. On the other hand, the observation we made in the previous paragraph tells us that a P system with non-cooperative rules cannot generate numbers belonging to a family which grows faster than what can be achieved with the right-hand sides of the rules, which means that no P system Π of depth $d + 1$ can generate the language L_d .

The proof of the previous theorem, combined with the observations made in Examples 1 and 2, leads us to the following corollary defining a depth-related hierarchy of polymorphic P systems with non-cooperative rules.

Corollary 4. $NOP_*^d(\text{polym}(n\text{coo})) \subset NOP_*^{d+1}(\text{polym}(n\text{coo}))$, for $d > 1$.

Theorem 4 also shows the relationship between *left*-polymorphic and general P systems. Indeed, Corollary 2 effectively states that the depth of any left-polymorphic P system is at most 3. This suggests the following assertion.

Corollary 5. $NOP_*(\text{lpolym}(n\text{coo})) \subset NOP_*(\text{polym}(n\text{coo}))$.

Establishing whether a similar inclusion is true for *right*-polymorphic P systems is an open problem.

8 Further Discussion and a Conclusion

In this paper we have explored some of the fundamental properties of the most basic class of polymorphic P systems introduced in [1]: polymorphic P systems

with non-cooperative rules and without any additional ingredients. We presented new special cases of such P systems (left- and right-polymorphic variants) and characterised the generative power of each of them, as well as of general polymorphic P systems. We have concluded the paper by bringing to light an *infinite* hierarchy of classes of polymorphic P systems with the base case being the usual non-polymorphic transition P systems.

The central conclusion of the paper is that, despite the apparent simplicity and restrictiveness of the definition, polymorphic P systems with non-cooperative rules and no additional ingredients are a very interesting object of study. We now directly proceed to formulating the questions which have been left unanswered in the paper.

The first question is concerned with the right-polymorphic variant of P systems. Just by looking at the definitions of left- and right-polymorphic P systems, it might seem that the generative power of both of them is strictly less than that of the most general version, and this is indeed the case for the left-polymorphic variant. The situation turns out to be much more complicated for *right*-polymorphic P systems: the author has no intuition whatsoever with respect to whether right-polymorphic P systems are less powerful than the general variant or not. On the one hand, the possibility to control left-hand sides of rules seems to increase the generative power of the model. On the other hand, all the proofs of all results concerning the upper bounds on the complexity of the languages produced by polymorphic P systems are *directly* applicable to the right-polymorphic case. Therefore, the fundamental question still to be answered in subsequent research is whether right-polymorphic P systems can generate all languages which can be produced by the general variant and if not, which are the languages separating the two classes.

A further question concerns the way in which right-polymorphic P systems are related to their left-polymorphic counterparts. We have seen that both of them can produce the number language $\{2^n \mid n \in \mathbb{N}\}$, but it is not clear whether the class of languages produced by the former family covers the one produced by the latter family. We suppose that the answer to this question is closely related to the reasoning exposed in the previous paragraph.

Finally, we would like to remember that we have only considered *the most restricted* version of polymorphic P systems in this paper. While adding ingredients to the model and/or allowing cooperative rules quickly renders the systems computationally complete [1] and thus (probably) less interesting for strictly theoretical research, applications of such models may present some interest. The fundamental feature of polymorphic P systems is that, in a way, they lie in between conventional models with static membrane structures, which require substantial design effort to attack practical problems, and the models with dynamic membrane structures, which often rely on *duplicating* computing units on the fly, together with their data [5]. This position, which we believe to be quite advantageous, may make polymorphic P systems a *practical* tool for solving certain real-world problems.

Acknowledgements Sergiu Ivanov gratefully acknowledges Artiom Alhazov for fruitful discussions.

References

1. Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic P systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 81–94. Springer Berlin Heidelberg, 2011.
2. Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) P systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer Berlin Heidelberg, 2007.
3. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.
4. Gheorghe Păun. *Membrane Computing: An Introduction*. Natural Computing Series Natural Computing. Springer, 2002.
5. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems with active membranes: Trading time for space. 10(1):167–182, March 2011.
6. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
7. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
8. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, Vol. 3: Beyond Words*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

Simulating Elementary Active Membranes With an Application to the P Conjecture*

Alberto Leporati, Luca Manzoni, Giancarlo Mauri,
Antonio E. Porreca, and 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

Abstract. The decision problems solved in polynomial time by P systems with elementary active membranes are known to include the class $\mathbf{P}^{\#\mathbf{P}}$. This consists of all the problems solved by polynomial-time deterministic Turing machines with polynomial-time counting oracles. In this paper we prove the reverse inclusion by simulating P systems with this kind of machines: this proves that the two complexity classes coincide, finally solving an open problem by Păun on the power of elementary division. The equivalence holds for both uniform and semi-uniform families of P systems, with or without membrane dissolution rules. Furthermore, the inclusion in $\mathbf{P}^{\#\mathbf{P}}$ also holds for “P conjecture systems” (with elementary division and dissolution but no charges), which improves the previously known upper bound \mathbf{PSPACE} .

1 Introduction

The computational power of P systems with elementary active membranes working in polynomial time was first investigated in [10]. The ability of P systems to exploit parallelism to perform multiple computations at the same time allows an efficient solution of \mathbf{NP} -complete problems. This feature was further exploited [8] to show that P systems with elementary active membranes can solve all $\mathbf{P}^{\mathbf{P}\mathbf{P}}$ problems (or $\mathbf{P}^{\#\mathbf{P}}$ problems, due to the equivalence of the two classes).

While all the previous results showed an ever increasing lower bound for the power of this class of P systems, the upper bound for their computational ability was proved to be \mathbf{PSPACE} [9]. Therefore, until now it was only known that this class is located between $\mathbf{P}^{\#\mathbf{P}}$ and \mathbf{PSPACE} . In this paper we show that the already known lower bound is, in fact, also an upper bound. This implies that non-elementary membrane division is necessary in order to solve \mathbf{PSPACE} -complete problems (unless $\mathbf{P}^{\#\mathbf{P}} = \mathbf{PSPACE}$), as conjectured by Sosík and Pérez-Jiménez and formulated as Problem B by Păun in 2005 [5]. This bound

* This work was partially supported by Università degli Studi di Milano-Bicocca, FA 2013: “Complessità computazionale in modelli di calcolo bioispirati: Sistemi a membrane e sistemi di reazioni”.

has also an interesting implication for the *P conjecture* [5, Problem F], stating that P systems with elementary division and dissolution but no charges can solve only problems in **P**. The previously known upper bound for the computational power of that class of P systems was also **PSPACE** but, since those systems are a weaker version of the ones studied here, the **P^{#P}** bound also applies to them.

The main idea behind the simulation of P systems with elementary active membranes by Turing machines with **#P** oracles is similar to one from [9]: we cannot store the entire configuration of the P system, since it can grow exponentially in time due to elementary membrane division. Therefore, instead of simulating directly the behaviour of the elementary membranes, we only simulate the interactions between them and their parent regions. Indeed, from the point of view of a non-elementary membrane, all the membranes it contains are just “black boxes” that absorb and release objects. We thus only store the configurations of non-elementary membranes and, when needed, we exploit a **#P** oracle to determine how many instances of each type of object are exchanged between elementary membranes and their parent regions. As will be clear in the following, while doing so we also need to take special care for send-in rules, since they require, in some sense, a partial knowledge of the parent’s multiset of objects, and conflicts between send-in rules competing for the same objects that can be applied to different membranes may be difficult to resolve. Thus, we have devised a way to provide a “centralised control” that allows to correctly apply the different send-in rules.

The paper is structured as follows. In Section 2 the basic definitions concerning P systems and the relevant complexity classes are briefly recalled. The simulation of P systems is described in Section 3; in particular, the three phases of the simulation algorithm requiring more computing power than a deterministic Turing machine working in polynomial time are detailed in Section 3.1 (movement of objects into elementary membranes), Section 3.2 (releasing objects from elementary membranes), and Section 3.3 (establishing if a membrane is elementary). The theorem stating the main result and the implications for the P conjecture are presented in Section 4. The paper is concluded with a summary of the results and some directions for future research in Section 5.

2 Basic Notions

We begin by recalling the basic definition of P systems with (elementary) active membranes [4].

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

- Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called objects;
- Λ is a finite set of labels for the membranes;
- μ is a membrane structure (i.e., a rooted unordered tree, usually represented by nested brackets) consisting of d membranes labelled by elements of Λ in a one-to-one way;

- w_{h_1}, \dots, w_{h_d} , with $h_1, \dots, h_d \in \Lambda$, are strings over Γ , describing the initial multisets of objects placed in the d regions of μ ;
- R is a finite set of rules.

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.

The rules in R are of the following types¹:

- (a) *Object evolution rules*, of the form $[a \rightarrow w]_h^\alpha$
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).
- (b) *Send-in communication rules*, of the form $a []_h^\alpha \rightarrow [b]_h^\beta$
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 becomes β .
- (c) *Send-out communication rules*, of the form $[a]_h^\alpha \rightarrow []_h^\beta b$
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 becomes β .
- (d) *Dissolution rules*, of the form $[a]_h^\alpha \rightarrow b$
They can be applied to a membrane labelled by h , having charge α and containing an occurrence of the object a ; the membrane is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b .
- (e) *Elementary division rules*, of the form $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$
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 of the multiset are replicated in both membranes.

Notice that “being an elementary membrane” is a *dynamic* property: even if a membrane originally contained other membranes, the dissolution of all of them makes the membrane elementary; if this happens, we also assume that any elementary division rules involving it become applicable, provided that its multiset and charge match the left-hand sides of the rules². Clearly, once a membrane is elementary it can never become non-elementary.

Since it is an essential technical detail in this paper, we carefully distinguish the concepts of *membrane* and *membrane label*, since membrane division allows

¹ The general definition of P systems with active membranes [4] also includes *non-elementary division rules* (type (f)), which are not used in this paper.

² The results of this paper, as well as the previous $\mathbf{P}^{\#\mathbf{P}}$ lower bound [8], continue to hold even if we assume that R can only contain elementary division rules for membranes that are *already* elementary in the initial configuration of the P system.

the creation of multiple membranes sharing the same label. In the rest of the paper, we use the expression “membrane h ” (singular) only when a single membrane labelled by h is guaranteed to exist, and refer to “membranes (labelled by) h ” (plural) otherwise.

Definition 2. *The instantaneous configuration of a membrane consists of its label h , its charge α , and the multiset w of objects it contains at a given time. It is denoted by $[w]_h^\alpha$. The (full) configuration \mathcal{C} of a P system Π at a given time is a rooted, unordered tree. The root is a node corresponding to the external environment of Π , and has a single subtree corresponding to the current membrane structure of Π . Furthermore, the root is labelled by the multiset located in the environment, and the remaining nodes by the configurations $[w]_h^\alpha$ of the corresponding membranes.*

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). Analogously, each membrane can only be subject to one communication, dissolution, or elementary division rule (types (b)–(e)) per computation step. 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 $\mathcal{C} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$ of configurations, 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 .

P systems can be used as language *recognisers* by employing two distinguished objects **yes** and **no**: we assume that all computations are halting, and that either object **yes** or object **no** (but not both) is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. All P systems in this paper are assumed to be confluent.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser P systems $\Pi = \{\Pi_x : x \in \Sigma^*\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^*$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [2].

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

- $F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n with a distinguished input membrane.
- $E(x) = w_x$, where w_x is a multiset encoding the specific input x .
- Finally, Π_x is simply Π_n with w_x added to its input membrane.

The family Π is said to be (polynomial-time) semi-uniform if there exists a single deterministic polynomial-time Turing machine H such that $H(x) = \Pi_x$ for each $x \in \Sigma^*$.

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 [2] for further details on the encoding of P systems.

Recall that the classes of decision problems solved by uniform and semi-uniform families of P systems working in polynomial time with evolution, communication, dissolution, and elementary division rules are denoted by $\mathbf{PMC}_{\mathcal{AM}(-n)}$ and $\mathbf{PMC}_{\mathcal{AM}(-n)}^*$, respectively. When dissolution is not allowed, the two corresponding classes are denoted by $\mathbf{PMC}_{\mathcal{AM}(-d,-n)}$ and $\mathbf{PMC}_{\mathcal{AM}(-d,-n)}^*$.

Finally, we recall the definitions of the complexity classes $\#\mathbf{P}$ and $\mathbf{P}\#\mathbf{P}$ (the latter being equivalent to $\mathbf{P}^{\mathbf{P}\mathbf{P}}$), and a notion of completeness for $\#\mathbf{P}$ [3].

Definition 4. The complexity class $\#\mathbf{P}$ consists of all the functions $f : \Sigma^* \rightarrow \mathbb{N}$, also called counting problems, with the following property: there exists a polynomial time nondeterministic Turing machine N such that, for each $x \in \Sigma^*$, the number of accepting computations of N on input x is exactly $f(x)$.

A function $f \in \#\mathbf{P}$ is said to be $\#\mathbf{P}$ -complete under parsimonious reductions if and only if for each $g \in \#\mathbf{P}$ there exists a polynomial-time reduction $r : \Sigma^* \rightarrow \Sigma^*$ such that $g(x) = f(r(x))$ for each $x \in \Sigma^*$.

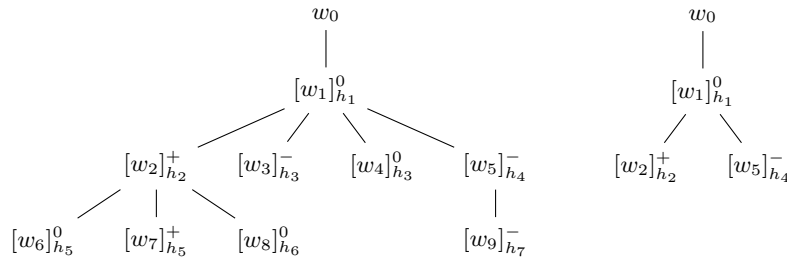


Fig. 1. A full configuration and a partial configuration for the same P system, where in the latter the configurations of all elementary membranes have been removed.

Definition 5. The complexity class $\mathbf{P}^{\#\mathbf{P}}$ consists of all decision problems (languages) recognisable in polynomial time by deterministic Turing machines with oracles for $\#\mathbf{P}$ functions. These are Turing machines M^f , with $f \in \#\mathbf{P}$, having a distinguished oracle tape and a query state such that, when M^f enters the query state, the string x on the oracle tape is immediately (i.e., at unitary time cost) replaced with the binary encoding of $f(x)$.

3 The Simulation Algorithm

Let $L \in \mathbf{PMC}_{\mathcal{AM}(-n)}^*$ be a language, and let $\mathbf{II} = \{II_x : x \in \Sigma^*\}$ be a semi-uniform family of confluent recogniser P systems with elementary active membranes deciding L in polynomial time. Let H be a polynomial-time deterministic Turing machine computing the mapping $x \mapsto II_x$.

A straightforward deterministic simulation of \mathbf{II} , which stores the entire configuration of a P system and applies the rules in a step-by-step fashion, in general incurs in an exponential slowdown, due to the exponential size of the configurations of the P systems, even when a binary encoding of the multisets is employed: indeed, by the ‘‘Milano Theorem’’ [10], the exponential space (and time) blowup of the simulation is due to the exponential number of elementary membranes created by division, and not to the number of objects itself. Here we simulate \mathbf{II} with a polynomial slowdown by means of a deterministic Turing machine with a $\#\mathbf{P}$ counting oracle.

We begin by observing that we do not need to *explicitly* store the configurations of the elementary membranes (either those that are elementary in the initial configuration, or those that become elementary during the computation), *as long as we can somehow keep track of the interactions between them and their parent regions*. Specifically, we need a way to track the objects that are released from the elementary membranes (via send-out or dissolution rules) and those that are absorbed by them (via send-in rules). We call *partial configuration* the portion of configuration of a P system that we store explicitly, in which we exclude the elementary membranes (unless, temporarily, in the very moment they become elementary and there is only a polynomial number of them) but include

```

1 construct the P system  $\Pi_x$  by simulating  $H$  on  $x$ 
2 let  $\mathcal{C}$  be the initial (full) configuration of  $\Pi_x$ 
3 for each time step  $t$  do
4   for each label  $h$  in  $\mathcal{C}$  do
5     if  $h$  denotes an elementary membrane then
6       remove its configuration from  $\mathcal{C}$ 
7     else
8       apply the rules to membrane  $h$ 
9       remove from membrane  $h$  the objects sent into
          elementary children membranes (see Algorithm 2)
10    for each label  $h'$  denoting elementary membranes do
11      add the output of all membranes
          labelled by  $h'$  to their parent multiset
12    if yes or no is found in the environment then
13      accept or reject accordingly

```

Algorithm 1. Simulation of a semi-uniform family Π of P systems with elementary active membranes on input x .

the environment surrounding the outermost membrane. An example of partial configuration is shown in Fig. 1.

Since the environment is where the result of any computation of a recogniser P system Π_x is ultimately decided, by the presence of an object **yes** or **no**, an up-to-date partial configuration is sufficient for a Turing machine to establish whether Π_x accepts or rejects. Furthermore, a partial configuration can be stored efficiently when the multisets are encoded in binary, since it consists of a polynomial number of regions containing an exponential number of objects [7].

The main technical result of this paper is a procedure for computing the final partial configuration of a P system in Π from the previous partial configurations and the initial full configuration, by using a counting oracle to reconstruct the interactions between the partial configuration and the elementary membranes. This simulation exploits the assumption of confluence of the P systems in Π when choosing the multiset of rules to apply at every time step. In principle, the simulation can be adapted to work with any deterministic way of choosing the rules but, in order to simplify the presentation, we give the following priorities:

Object evolution rules have the highest priority, followed by send-in rules, followed by the remaining types of rule. Inside these three classes, the priorities are given by any fixed total order.

Algorithm 1 provides an overview of the simulation; the algorithm takes a string $x \in \Sigma^*$ as input. First of all, it simulates the machine H that provides the uniformity condition for Π on input x , thus obtaining a description of the P system Π_x (line 1). It also stores the initial configuration of Π as a suitable labelled tree data structure (line 2).

The main loop (lines 3–13) is repeated for each time step of Π_x , and consists of three main phases. First, the algorithm iterates (lines 4–9) through all membrane labels in the current partial configuration \mathcal{C} . The configurations of membranes that have become elementary during the previous computation step are removed from \mathcal{C} (lines 5–6), as described in Section 3.3. Instead, the rules are applied normally [10] to non-elementary membranes (line 8), and the objects that are sent into their children elementary membranes are also removed from them (line 9); the latter operation will be detailed in Section 3.1. In the second phase the algorithm computes the output of the elementary membranes and adds it to their parent membranes (lines 10–11). This phase will be detailed in Section 3.2. Finally, in the third phase (lines 12–13) the algorithm checks whether the object **yes** (resp., **no**) was sent out from the outermost membrane in the current computation step of Π_x ; if this is the case, the simulation halts by accepting (resp., rejecting) x .

3.1 Moving Objects *into* Elementary Membranes

Line 9 of the simulation algorithm requires the removal from a specific non-elementary membrane h of the objects sent into its elementary children membranes. While the current configuration of h is known, as it is stored in the partial configuration \mathcal{C} , the configuration of its children membranes is not stored.

We solve this problem by nondeterministically simulating each children membrane of h having label h' ; each computation keeps track of just a single membrane h' among those obtained by division, starting from the moment the initial (unique) membrane h' became elementary, up to the current time step of the P system. Hence, whenever a membrane becomes elementary (a fact discovered at line 5 of the simulation algorithm), before removing it from \mathcal{C} the algorithm stores its configuration, together with the current time step t , in an auxiliary table of polynomial size, indexed by membrane labels.

We are only interested in simulating the *internal* configuration of elementary membranes (i.e., the multiset it contains and its charge), without keeping track of any objects released from them either by send-out or dissolution rules. Hence, the only problematic rules are those of type send-in, since multiple membranes sharing the same parent compete for the same objects, but cannot coordinate because each membrane is simulated by a different computation; recall that distinct computations of a nondeterministic Turing machine do not communicate. In order to solve the conflicts, a table $\text{apply}[r, t]$ can be precomputed, associating each send-in rule $r = a []_h^\alpha \rightarrow [b]_h^\beta$ and time step t with the set of individual membranes with label h that must apply r at time t .

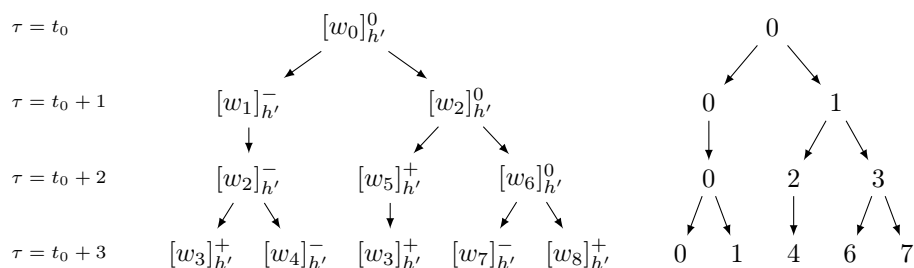
Computing the entries of the table apply first requires us to name each individual membrane among those sharing the same label. In order to do so, we exploit a solution already proposed by Sosík and Rodríguez-Patón [9], attaching to each elementary membrane an identifier computed as follows:

- When membrane h becomes elementary, its identifier is $id = 0$.
- If a membrane has identifier id at time τ and no division occurs at that time, its new identifier at time $\tau + 1$ is $2 \times id$.

- If a membrane has identifier id at time τ and a division occurs at that time, the new identifiers of the two resulting membranes at time $\tau + 1$ are $2 \times id$ and $2 \times id + 1$, respectively.

This ensures that, at time t , each membrane with label h has a unique identifier in the range $[0, 2^t - 1]$ (although not all identifiers in that range need to correspond to actual membranes).

Example 1. The procedure that assigns unique identifiers to elementary membranes sharing the same label can be represented graphically. On the left, we show a tree of membrane divisions (not to be confused with a configuration tree, as illustrated in Fig. 1): time goes downward, starting from the moment t_0 when the membrane became elementary, each level representing a time step; notice that not all membranes necessarily divide at each step. On the right, we show the identifiers assigned to the membranes:



Notice how all identifiers differ, even for membranes having the same configuration: for example, the two copies of $[w_3]_{h'}^+$ at time $\tau = t_0 + 3$ have identifiers 0 and 4. Furthermore, not all identifiers between 0 and $2^t - 1$ correspond to a membrane: here 2, 3, and 5 do not correspond to any membrane.

Naively storing arbitrary subsets of $[0, 2^t - 1]$ as entries of the table **apply** requires an exponential amount of space with respect to t . We can, however, exploit the hypothesis of confluence of the family **II** and choose, among all possible computations of the simulated P system, the computation where each send-in rule $r = a []_h^\alpha \rightarrow [b]_h^\beta$ is applied, at each step t , to all (and only the) membranes h having charge α whose identifiers are included in an interval, to be computed as described below. Distinct intervals involving the same membrane label h may overlap only for rules r_1, r_2 having a different charge on the left-hand side, as shown in the following example.

Example 2. Consider the P system of Example 1 at time $t = 3$, and suppose the following rules (sorted by priority) are associated to the membrane label h' :

$$r_1 = a []_{h'}^- \rightarrow [c]_{h'}^+ \quad r_2 = a []_{h'}^+ \rightarrow [d]_{h'}^+ \quad r_3 = b []_{h'}^+ \rightarrow [e]_{h'}^0$$

Furthermore, suppose the parent membrane h of h' contains exactly 3 instances of object a and 4 instances of b . Then, one of the possible choices of intervals of

```

e1 set  $id := 0$ 
e2 for each time step  $\tau \in \{t_0, \dots, t-1\}$  do
e3   set  $newid := 2 \times id$ 
e4   for each applicable evolution rule  $r \in R$  involving  $h'$  do
e5     apply  $r$  as many times as possible
e6   for each non-evolution rule  $r$  applicable to  $h'$  do
e7     if  $r = [a]_{h'}^\alpha \rightarrow [ ]_{h'}^\beta b$  then
e8       remove an instance of  $a$  from the membrane
e9       change the charge of the membrane to  $\beta$ 
e10    else if  $r = [a]_{h'}^\alpha \rightarrow b$  then
e11      reject
e12    else if  $r = [a]_{h'}^\alpha \rightarrow [b_0]_{h'}^{\beta_0} [b_1]_{h'}^{\beta_1}$  then
e13      nondeterministically guess a bit  $i$ 
e14      set  $newid := newid + i$ 
e15      rewrite an instance of  $a$  to  $b_i$ 
e16      change the charge of the membrane to  $\beta_i$ 
e17    else if  $r = a [ ]_{h'}^\alpha \rightarrow [b]_{h'}^\beta$  and  $\text{left}[r, \tau] \leq id \leq \text{right}[r, \tau]$  then
e18      add an instance of  $b$  to the membrane
e19      change the charge of the membrane to  $\beta$ 
e20   set  $id := newid$ 

```

Algorithm 2. Nondeterministic simulation of steps t_0 to $t-1$ of an elementary membrane h' . Here t_0 is the time when membrane h' became elementary.

identifiers, which is consistent with the previous discussion, is

$$\text{apply}[r_1, t] = [0, 7] \quad \text{apply}[r_2, t] = [0, 3] \quad \text{apply}[r_3, t] = [4, 7]$$

The interval $\text{apply}[r_1, t]$ contains exactly 2 identifiers of membranes h' having negative charge (the maximum number of instances of a that can be sent in by means of rule r_1); the interval $\text{apply}[r_2, t]$ contains only 1 identifier of membranes h' with positive charge (since only one copy of a remains that is not used up by rule r_1); finally, the interval $\text{apply}[r_3, t]$ contains the remaining 2 identifiers of membranes h' with positive charge, leaving two instances of b unused by send-in rules. Notice that the interval $\text{apply}[r_1, t]$ overlaps with the others, since rule r_1 is applied to membranes h' having different charge than those where r_2 and r_3 may be applied, while $\text{apply}[r_2, t]$ and $\text{apply}[r_3, t]$ are necessarily disjoint.

In order to save space, we can replace each interval $\text{apply}[r, t]$ with its bounds $\text{left}[r, t]$ and $\text{right}[r, t]$. Storing the intervals of identifiers then requires only two binary numbers per each original entry $\text{apply}[r, t]$; hence, the information contained in the table can now be stored in polynomial space.

The pseudocode for the nondeterministic simulation of the elementary membranes labelled by h' , starting from the time t_0 when h' became elementary and up to time $t-1$ (i.e., leaving out the last computation step), is shown as

Algorithm 2. This is a straightforward simulation algorithm for P systems, as already described in the literature [10, 7], except for a few key differences:

- the simulation aborts by rejecting if the membrane dissolves within step $t - 1$ (lines e10–e11);
- when simulating an elementary division (lines e12–e16) the algorithm performs a nondeterministic choice between the two membranes resulting from the division, and continues to simulate only one of them;
- before simulating a send-in rule (lines e17–e19), the algorithm checks whether the identifier of the membrane being simulated belongs to the range of identifiers of membranes applying that rule at time τ .

Notice that the loop of lines e6–e19 applies at most one non-evolution rule, as required by the semantics of P systems with active membranes.

Let $r = a []_{h'}^\alpha \rightarrow [b]_{h'}^\beta$ be a send-in rule. We can simulate a further computation step of each membrane labelled by h' , halting by accepting if rule r is actually applicable (i.e., the current membrane h' has charge α , and its identifier is in the correct range), and rejecting otherwise. As a consequence, Algorithm 2 with this additional step has as many accepting computations as the number of membranes having label h' where rule r is applied at time t . This proves the following:

Lemma 1. *Suppose we are given the configuration $[w]_{h'}^\alpha$ of an elementary membrane h' , a set of rules, two time steps t_0 (corresponding to configuration $[w]_{h'}^\alpha$) and t expressed in unary notation³, two tables **left** and **right** as described above, and a specific send-in communication rule $r = a []_{h'}^\alpha \rightarrow [b]_{h'}^\beta \in R$. Then, counting the number of applications of rule r at time t is in $\#\mathbf{P}$. \square*

Remark 1. Counting the number of satisfying assignments for a Boolean formula in 3CNF (a $\#\mathbf{P}$ -complete problem) can be reduced in polynomial time to counting the number of send-in rules applied during the “counting” step by the P systems solving the THRESHOLD-3SAT problem described in [6]. This requires setting the threshold to 2^m , where m is the number of variables. Hence, the counting problem of Lemma 1 is actually $\#\mathbf{P}$ -complete.

Line 9 of Algorithm 1 can thus be expanded as Algorithm 3, where the entries of **left** and **right** are also computed. The number of objects sent in from h to its children membranes h' by means of a rule $r = a []_{h'}^\alpha \rightarrow [b]_{h'}^\beta$ can be computed (line 9.8) as the minimum k between the number of instances of a contained in h and the number m of membranes h' having charge α which have not yet been assigned a send-in rule. The initial value of m is thus the number of membranes h' having identifier at least $\mathbf{left}[r, t]$ and charge α . The membranes h' having charge α and an identifier smaller than $\mathbf{left}[r, t]$ have already been assigned to a different interval, and will apply a different send-in rule. The algorithm can then remove k copies of a from h (line 9.9). The value of $\mathbf{right}[r, t]$ must then be updated, in

³ Expressing the number of time steps t in unary is necessary for the problem to be in $\#\mathbf{P}$, otherwise the value t might be exponentially larger than its representation, thus increasing the complexity of the problem.

```

9.1 for each elementary membrane label  $h'$  contained in  $h$  do
9.2   for each charge  $\alpha \in \{+, 0, -\}$  do
9.3     set  $\ell := 0$ 
9.4     for each rule  $r = a [ ]_{h'}^\alpha \rightarrow [b]_{h'}^\beta$  in  $R$  do
9.5       set  $\text{left}[r, t] := \ell$ 
9.6       set  $\text{right}[r, t] := 2^t - 1$ 
9.7       let  $m$  be the number of membranes with label  $h'$  and
            $\text{left}[r, t] \leq id \leq \text{right}[r, t]$  where  $r$  is applicable at time  $t$ 
9.8       set  $k := \min\{m, \text{number of instances of } a \text{ in } h\}$ 
9.9       remove  $k$  instances of  $a$  from  $h$ 
9.10      while  $m \neq k$  do
9.11        update  $\text{right}[r, t]$  by binary search
9.12        recompute  $m$ 
9.13      set  $\ell := \text{right}[r, t] + 1$ 

```

Algorithm 3. Removing from a non-elementary membrane h the objects sent into its elementary children membranes (Algorithm 1, line 9).

order to ensure that exactly k membranes with label h' apply r at time t . A correct value for $\text{right}[r, t]$ can be determined by performing a binary search in the interval $[\text{left}[r, t], 2^t - 1]$ while recomputing the value of m (line 9.12) as in Lemma 1.

3.2 Moving Objects *from* Elementary Membranes

Line 11 of Algorithm 1 deals with communication in the opposite direction with respect to line 9, i.e., from the elementary membranes towards their parent.

For each label h' denoting an elementary membrane and for each object type a , the number of instances of a released from membranes with label h' at time t can be determined by first simulating these membranes up to time $t - 1$ by using Algorithm 2; the last time step is then simulated as follows:

- If a rule sending out a is applied, the computation accepts.
- If the membrane dissolves, the algorithm accepts as many times as the number k of instances of a in the simulated membrane. This requires “forking” k accepting computations, which in nondeterministic Turing machines corresponds to first nondeterministically guessing a number between 1 and k , and then accepting. This can be performed in polynomial time even if k is exponential, since the number of bits of k is polynomial.
- Otherwise, the computation rejects.

The number of accepting computations of the algorithm is the number of instances of a to be added to the parent of h' . This proves the following:

Lemma 2. *Suppose we are given the configuration $[w]_{h'}^\alpha$ of an elementary membrane h' , a set of rules, two time steps t_0 (corresponding to configuration $[w]_{h'}^\alpha$)*

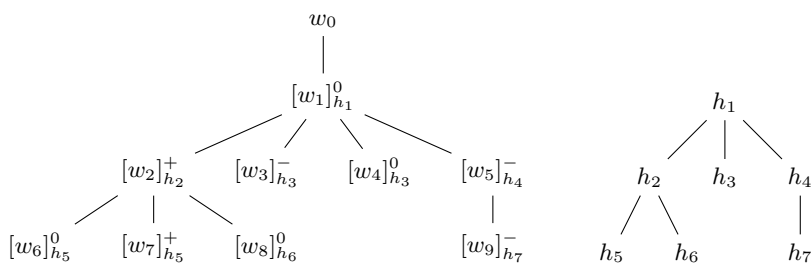


Fig. 2. A full configuration and the tree \mathcal{T} containing the information about the inclusions among its membranes, expressed in terms of labels only.

and t expressed in unary notation, two tables left and right as described above, and an object type a . Then, counting the number of instances of object a released (via send-out or dissolution rules) from membranes with label h' at time t is in $\#\mathbf{P}$. \square

Remark 2. We can reduce the assignment counting problem of Remark 1 to counting the number of objects sent out by elementary membranes. Indeed, during the “checking” phase of the aforementioned algorithm for THRESHOLD-3SAT [6], all membranes containing a satisfying assignment simultaneously send out a specific object. Hence, the counting problem of Lemma 2 is also $\#\mathbf{P}$ -complete.

3.3 Deciding if a Membrane is Elementary

In order to decide (line 5 of Algorithm 1) if a label h belongs to an elementary membrane at time t (recall that membranes sharing the same label are either all elementary and share the same parent, or there is only one of them) without keeping them in the partial configuration \mathcal{C} , we can use an auxiliary data structure. We employ a rooted, unordered tree \mathcal{T} whose nodes are labels from Λ , i.e., no repeated labels appear in \mathcal{T} . This tree represents the inclusion relation between membranes before each simulated step, and it is initialised before entering the main loop of the simulation (lines 3–13). As an example, Fig. 2 shows a full configuration and the corresponding inclusion tree between membranes labels. This data structure is updated in two different steps of Algorithm 1: at line 8, when non-elementary membranes dissolve, and after the execution of line 5 (as described below).

Then, for each label h appearing in the partial configuration \mathcal{C} (line 4), either it is associated to an internal node in \mathcal{C} , and in that case it is not elementary, or it is a leaf in \mathcal{C} , and then the node h has leaf children in \mathcal{T} . In that case, since we do not keep track of the elementary membranes in partial configurations, the algorithm needs to explicitly check, for each child membrane label h' of h , whether there does actually still exist at least one membrane with label h' .

Once again, we employ Algorithm 2 to simulate the elementary membranes with label h' up to time step $t - 1$; if the algorithm does not have already

rejected by then, this means that the simulated membrane still exists, and the computation must accept. In this case there is no need to count the accepting computations, but only to check whether there exists at least one. We can thus state the following:

Lemma 3. *Suppose we are given the configuration $[w]_{h'}^\alpha$ of an elementary membrane h' , a set of rules, two time steps t_0 (corresponding to configuration $[w]_{h'}^\alpha$) and t expressed in unary notation, and two tables **left** and **right** as described above. Then, deciding whether there exists a membrane with label h' at time t is in **NP**. \square*

Remark 3. The most common solutions to **NP**-complete problems using **P** systems with elementary active membranes (such as [10]) first generate one membrane per candidate solution, then check all of them for validity. By dissolving all membranes containing invalid solutions, we can easily show that the decision problem of Lemma 3 is actually **NP**-complete by reduction from any other **NP**-complete problem.

4 A Characterisation of $\mathbf{P}\#\mathbf{P}$

We are now able to analyse the complexity of the simulation algorithm.

Lemma 4. *Algorithm 1 runs in polynomial time on a deterministic Turing machine with access to an oracle for a $\#\mathbf{P}$ -complete function.*

Proof. Lines 1 and 2 can be executed in polynomial time due to the assumption of semi-uniformity of the family **II**. The loop of line 3 simulates a polynomial number of time steps, and those of lines 4 and 10 iterate over sets of membrane labels. Hence, lines 5–9 and 11–13 are executed a polynomial number of times.

Line 6 simply consists in removing a node from a tree data structure, line 8 in the application of rules to a polynomial number of membranes, which can be performed in polynomial time [10, 7], and lines 12–13 in a membership test.

Let $f: \Sigma^* \rightarrow \mathbb{N}$ be $\#\mathbf{P}$ -complete under parsimonious reductions (e.g., $\#\text{SAT}$). Then, the counting problems of Lemmata 1 and 2 and the decision problem of Lemma 3 can be reduced in polynomial time to evaluating f . Suppose we have access to an oracle for f .

The implementation of line 5 described in Section 3.3 employs the auxiliary tree data structure \mathcal{T} (which can be scanned and updated in polynomial time), and can ascertain the existence of children membranes of h by performing a reduction and a query to the oracle for f at most once per label.

Moving objects into elementary membranes (line 9), as described in Section 3.1, executes Algorithm 3. This performs a polynomial number of iterations of lines 9.5–9.13. Lines 9.7 and 9.12 are reductions followed by an oracle query for f . The loop of lines 9.10–9.12 is executed a logarithmic number of times with respect to the exponential number of identifiers (i.e., a polynomial number).

Finally, line 11 of Algorithm 1, whose implementation is described in Section 3.2, computes the output of the elementary membranes. This also consists in

performing a reduction and an oracle query to f for each elementary membrane label and each object type. The statement of the lemma follows. \square

As a consequence, we obtain the equivalence of several complexity classes for P systems with elementary active membranes to $\mathbf{P\#P}$.

Theorem 1. *The following equalities hold:*

$$\mathbf{PMC}_{\mathcal{AM}(-d,-n)}^{[*]} = \mathbf{PMC}_{\mathcal{AM}(-n)}^{[*]} = \mathbf{P\#P}$$

where $[*]$ denotes optional semi-uniformity instead of uniformity.

Proof. The inclusions of the following diagram hold:

$$\begin{array}{ccc} & \mathbf{PMC}_{\mathcal{AM}(-d,-n)}^* & \\ & \subseteq & \\ \mathbf{P\#P} \subseteq \mathbf{PMC}_{\mathcal{AM}(-d,-n)} & & \subseteq \mathbf{PMC}_{\mathcal{AM}(-n)}^* \subseteq \mathbf{P\#P} \\ & \subseteq & \\ & \mathbf{PMC}_{\mathcal{AM}(-n)} & \end{array}$$

Indeed, uniformity implies semi-uniformity, and adding dissolution rules does not decrease the power of P systems. Furthermore, P systems with restricted (without dissolution) elementary active membranes are able to efficiently simulate polynomial-time deterministic Turing machines with counting oracles [8]. The last inclusion $\mathbf{PMC}_{\mathcal{AM}(-n)}^* \subseteq \mathbf{P\#P}$ follows from Lemma 4. \square

4.1 Consequences for the P Conjecture

In P systems without charges, the presence of dissolution rules becomes important: without them, only problems in \mathbf{P} may be solved in polynomial time [1]. The P conjecture [5, Problem F] claims that not even elementary division together with dissolution rules can break this barrier, although the tightest known upper bound up to now was \mathbf{PSPACE} [9].

P systems without charges are equivalent to P systems *with* charges where the membranes always remain neutral (i.e., no rule ever changes a membrane charge). Hence, the class of problems solved in polynomial time by the former model without non-elementary division rules, denoted by $\mathbf{PMC}_{\mathcal{AM}^0(-n)}$, is trivially included in $\mathbf{PMC}_{\mathcal{AM}(-n)}$. The corresponding inclusion also holds in the semi-uniform case. As a consequence, Theorem 1 implies the following improved upper bound for the P conjecture:

Corollary 1. $\mathbf{PMC}_{\mathcal{AM}^0(-n)}^{[*]} \subseteq \mathbf{P\#P}$. \square

5 Conclusions

In this paper we have presented a simulation of polynomial-time semi-uniform families of P systems with elementary active membranes, characterising the complexity class $\mathbf{PMC}_{\mathcal{AM}(-n)}^*$, by means of deterministic Turing machines working in

polynomial time with access to a $\#\mathbf{P}$ oracle. This simulation and the previously known lower bound [8] complete the characterisation of this complexity class, as well as those obtained by requiring uniformity instead of semi-uniformity or disallowing dissolution rules: all these classes coincide with $\mathbf{P}\#\mathbf{P}$. This result is also interesting because it represents an improvement of the upper bound of the computational power for the class of P systems involved in the P conjecture. We hope that this step will help in the search for a solution to the conjecture.

In the future we plan to investigate the computational power of P systems with active membranes in which non-elementary division is allowed but limited to membranes of a certain depth. While unrestricted non-elementary division increases the computational power to \mathbf{PSPACE} , we conjecture that limited division can generate a hierarchy of complexity classes. Furthermore, we plan to investigate the computational power of non-confluent P systems with active membranes using division rules. Currently, no upper bound tighter than \mathbf{NEXP} is known for these classes of P systems.

References

1. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Nuñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* 83(7), 593–611 (2006)
2. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. *Natural Computing* 10(1), 613–632 (2011)
3. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley (1993)
4. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics* 6(1), 75–90 (2001)
5. Păun, Gh.: Further twenty six open problems in membrane computing. In: Gutiérrez-Naranjo, M.A., Riscos-Nuñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) *Proceedings of the Third Brainstorming Week on Membrane Computing*. pp. 249–262. Fénix Editora (2005)
6. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Elementary active membranes have the power of counting. *International Journal of Natural Computing Research* 2(3), 329–342 (2011)
7. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science* 22(1), 65–73 (2011)
8. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems simulating oracle computations. In: Gheorghe, M., Păun, Gh., Salomaa, A., Rozenberg, G., Verlan, S. (eds.) *Membrane Computing, 12th International Conference, CMC 2011, Lecture Notes in Computer Science*, vol. 7184, pp. 346–358. Springer (2012)
9. Sosik, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences* 73(1), 137–152 (2007)
10. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference*, pp. 289–301. Springer (2001)

Spiking Neural P Systems with Cooperating Rules

Venkata Padmavati Metta¹, Srinivasan Raghuraman², and
Kamala Krithivasan²

¹ Institute of Computer Science and Research Institute of the IT4Innovations Centre
of Excellence, Silesian University in Opava, Czech Republic

² Indian Institute of Technology, Chennai, India
vmetta@gmail.com, srini131293@gmail.com, kamala@iitm.ac.in

Abstract. The concept of cooperation and distribution as known from grammar systems is introduced to spiking neural P systems (in short, SN P systems) in which each neuron has a finite number of sets (called *components*) of rules. During computations, at each step only one of the components can be active for the whole system and one of the enabled rules from this active component of each neuron fires. The switching between the components occurs under different cooperation strategies. This paper considers the terminating mode, in which the switching occurs when no rule is enabled in the active component of any neuron in the system. By introducing this new mechanism, the computational power of asynchronous and sequential SN P systems with standard rules is investigated. The results are that asynchronous standard SN P systems with two components and strongly sequential unbounded SN P systems with two components are Turing complete.

1 Introduction

Cooperative grammar systems were introduced by Meersman and Rozenberg in [6], in the context of two-level grammars. The systematic study of cooperating distributed (for short, CD) grammar systems was initiated by Csuhaĵ-Varjú and Dassow in [2], where productions are distributed over a finite number of sets, called components. These components cooperate during the derivation process by applying productions on a common sentential form; following some fixed cooperation protocol.

The concept of cooperation and distribution as known from CD grammar systems is introduced to spiking neural P systems. Spiking neural P systems [5] are parallel and distributed computing models inspired by the neurophysiological behaviour of neurons sending electrical pulses of identical voltages called spikes to the neighbouring neurons through synapses. An SN P system is represented as a directed graph where nodes correspond to the neurons having spiking rules and forgetting rules. The rules involve the spikes present in the neuron in the form of occurrences of a symbol a . The arcs indicate the synapses among the neurons. The spiking rules are of the form $E / a^r \rightarrow a$ and are used only if

the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$, where $L(E)$ is the language represented by regular expression E . In this case a^r number of spikes are consumed and one spike is sent out. When neuron σ_i sends a spike, it is replicated in such a way that one spike is immediately sent to all neurons j such that $(i, j) \in \text{syn}$, where syn is the set of arcs between the neurons. The transmission of spikes takes no time, the spike will be available in neuron j in the next step. The forgetting rules are of the form $a^s \rightarrow \lambda$ and are applied only if the neuron contains exactly a^s spikes. The rule simply removes s spikes. For all forgetting rules, s must not be the member of $L(E)$ for any firing rule within the same neuron.

A rule is *bounded* if it is of the form $a^i/a^j \rightarrow a$, where $1 \leq j \leq i$, or of the form $a^k \rightarrow \lambda$, where $k \geq 1$. A neuron is bounded if it contains only bounded rules. A rule is called *unbounded* if it is of the form $a^c(a^i)^*/a^j \rightarrow a$, where $c \geq 0$, $i \geq 1$, $j \geq 1$. A neuron is unbounded if it contains only unbounded rules. A neuron is *general* if it contains both bounded and unbounded rules. An SN P system is bounded if all the neurons in the system are bounded. It is unbounded if it has bounded and unbounded neurons. Finally, an SN P system is general if it has general neurons (i.e., it contains at least one neuron which has both bounded and unbounded rules).

The usual SN P systems are synchronous (a global clock is assumed) and work in a maximally parallel manner, in the sense that all neurons that are fireable must fire. However, in any neuron, at most one rule is allowed to fire. One neuron is designated as the output neuron of the system and its spikes can exit into the environment, thus producing a spike train. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and a set of numbers, obtained by considering the total number of spikes emitted by the output neuron until the system halts. Two main types of results were obtained for synchronous SN P systems using standard rules (producing one spike): computational completeness in the case when no bound was imposed on the number of spikes present in the system, and a characterization of semi-linear sets of numbers in the case when a bound was imposed [5].

This paper introduces spiking neural P system with cooperating rules where each neuron has a finite number of sets of spiking and forgetting rules. Each set is called a component which can be empty. At any step or during a sequence of steps (depending on the mode of application) only one of the components is active for the whole system and only one of the enabled rules from this component of each neuron can fire during that step. After that another (not necessarily different) component of each neuron becomes active. The way of passing active control is called a protocol. Similar to CD grammar systems, series of cooperation protocols among the components in neurons of an SN P system can be considered, where for example any component, once started, has to perform exactly k , at most k , at least k , $k \geq 1$ or an arbitrary number of transition steps. In the so-called terminating mode, a component may stop working if and only if none of the rules

in that component of any neuron is applicable. In any case, the selection of the next active component is non-deterministic and only one component generates the output at a step, other components wait for receiving control.

This paper considers asynchronous SN P systems [1], where in any step, a neuron can apply or not apply its rules which are enabled by the number of spikes it contains (further spikes can come, thus changing the rules enabled in the next step). Because the time between two firings of the output neuron is now irrelevant, the result of a computation is the number of spikes sent out by the system, not the distance between certain spikes leaving the system. It was proved that such asynchronous SN P systems with extended rules are equivalent to Turing machines (as generators of sets of natural numbers) but universality of such systems with standard rules is still an open problem. The additional non-determinism introduced in the functioning of the system by the non-synchronization has more computing power in the case of using two components. That is, two component SN P systems with standard rules working asynchronously are equivalent to the Turing machines (interpreted as generators of sets of (vectors of) numbers).

The paper also considers sequential SN P systems in which, at every step of the computation, if there is at least one neuron with at least one rule that is fireable, we only allow one such neuron and one such rule (both nondeterministically chosen) to be fired. Here, not every step has at least one neuron with a fireable rule. (Thus, the system might be dormant until a rule becomes fireable. However, the clock will keep on ticking.) The sequential unbounded as well as general SN P systems are proved to be universal [4]. A system is strongly sequential, if at every step, there is at least one neuron with a fireable rule. It is shown that strongly sequential general SN P systems are universal but strongly sequential unbounded SN P systems are not universal [4]. In this paper, we also prove that strongly sequential unbounded SN P systems with two components are universal.

The paper is organized as follows. In the next section, register machines are defined. SN P systems with cooperating rules are introduced in Section 3. The universality of asynchronous two component SN P systems with standard rules is proved in Section 4 and that of strongly sequential SN P systems with standard unbounded neurons is proved in Section 5.

2 Prerequisites

We assume the reader to be familiar with formal language theory, CD grammar systems and membrane computing. The reader can find details about them in [10], [3], [9] etc.

The family of Turing computable sets of natural numbers is denoted by NRE (the notation comes from the fact that these numbers are the length sets of recursively enumerable languages). The family of NRE is also the family of sets of numbers generated/recognized by register machines. For the universality proofs in this paper, we use the characterization of NRE by means of register

machines [7]. Such a device - in the non-deterministic version - is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labelling an *ADD* instruction), l_h is the halt label (assigned to instruction *HALT*), and I is the set of instructions; each label from H labels only one instruction from I , thus precisely identifying it.

The labelled instructions are of the following forms:

1. $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),
2. $l_i : (SUB(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j , otherwise go to the instruction with label l_k),
3. $l_h : HALT$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers). If we reach the halt instruction, then the number n present in register 1 (the registers are numbered from 1 to m) at that time is said to be generated by M . It is known (e.g., see, [7]) that register machines generate all sets of numbers which are Turing computable.

A register machine can also accept a set of numbers: a number n is accepted by M if, starting with n in register 1 and all other registers empty, the computation eventually halts (without loss of generality, we may assume that in the halting configuration all registers are empty). Deterministic register machines (i.e., with *ADD* instructions of the form $l_i : (ADD(r), l_j)$ working in the accepting mode are known to be equivalent to Turing machines.

It is also possible to consider register machines producing sets of vectors of natural numbers. In this case a distinguished set of v registers (for some $v \geq 1$) are designated as the output registers. A v -tuple $(l_1, l_2, \dots, l_v) \in N^v$ is generated if M eventually halts and the contents of the output registers are l_1, l_2, \dots, l_v respectively.

Without loss of generality we may assume that in the halting configuration all the registers, except the output ones, are empty. We also assume (without loss of generality) that the output registers are non-decreasing and their contents is only incremented (i.e., the output registers are never the subject of *SUB* instructions, but only of *ADD* instructions).

We will refer to a register machine with v -output registers (the other registers are auxiliary registers) as a v -output register machine. It is well known that a set S of v -tuples of numbers is generated by a v -output register machine if and only if S is recursively enumerable. When dealing with vectors of numbers, hence with the Parikh images of languages (or with sets of vectors generated/recognized by register machines), we write *PsRE*.

3 Spiking Neural P Systems with Cooperating Rules

We pass on now to introducing SN P systems with cooperating rules investigated in this paper.

Definition 1. [SN P system with cooperating rules] An SN P system with cooperating rules is an SN P system of degree $m \geq 1$ with $p \geq 1$ components, of the form

$$\Pi = (O, \Sigma, \sigma_1, \sigma_2, \sigma_3, \dots, \sigma_m, \text{syn}, \text{out}), \text{ where}$$

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\Sigma = \{1, 2, \dots, p\}$ is the label alphabet for components;
3. $\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_m$ are neurons, of the form

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

where

- (a) $n_i \geq 0$ is the *initial number of spikes* contained in the neuron;
- (b) $R_i = \cup_{l \in \Sigma} R_{il}$, where each R_{il} , $1 \leq l \leq p$, is a set (can be empty) of rules representing a component l in σ_i having rules of the following two forms:
 - i. $E/a^r \rightarrow a$, where E is a regular expression over O , $r \geq 1$ (if $L(E) = a^r$, then we simply write $a^r \rightarrow a$);
 - ii. $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \rightarrow a$ of type i. from R_{il} ;
4. $\text{syn} \subseteq \{1, 2, 3, \dots, m\} \times \{1, 2, 3, \dots, m\}$ with $(i, i) \notin \text{syn}$ for $1 \leq i \leq m$ (*synapses* among cells);
5. $\text{out} \in \{1, 2, 3, \dots, m\}$ indicates the *output* neuron.

Because we do not need the delay between firing and spiking (i.e., rules of the form $E/a^r \rightarrow a; d$, with $d \geq 1$) as well as extended rules (i.e., rules of the form $E/a^r \rightarrow a^q$, with $q \geq 1$) in the proofs below, we do not consider these features in the definition, but such rules can be introduced in the usual way.

The rules of the type $E/a^r \rightarrow a$ are spiking rules, and can be applied only if the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$. When neuron σ_i spikes, its spike is replicated in such a way that one spike is sent immediately to all neurons σ_j such that $(i, j) \in \text{syn}$. The rules of type $a^s \rightarrow \lambda$ are forgetting rules; s spikes are simply removed (“forgotten”) when applying such a rule. Like in the case of spiking rules, the left-hand side of a forgetting rule must “cover” the contents of the neuron, that is, $a^s \rightarrow \lambda$ is applied only if the neuron contains exactly s spikes. For simplicity, in the graphical representation of the system, the rules in the component l of neuron σ_i are prefixed with l and the components inside the neuron is separated by lines.

As defined above, each component of the neurons can contain several rules. More precisely, it is allowed to have two spiking rules $E_1/a^{r_1} \rightarrow a$ and $E_2/a^{r_2} \rightarrow a$ with $L(E_1) \cap L(E_2) \neq \emptyset$ in the same component. This leads to a non-deterministic way of using the rules and we cannot avoid the non-determinism (deterministic systems will compute only singleton sets).

The *configuration* of an SN P system is described by the number of spikes in each neuron. Thus, the initial configuration of the system is described as $\mathcal{C}_0 = \langle n_1, n_2, n_3, \dots, n_m \rangle$.

The SN P system is synchronized by means of a global clock and works in a locally sequential and globally maximal manner with one component active at a step for the whole system. That is, the working is sequential at the level of each neuron. In each neuron, at each step, if there is more than one rule enabled from the active component by its current contents, then only one of them (chosen non-deterministically) can fire. But still, the system as a whole evolves in a parallel and synchronising way, as in each step, all the neurons (that have an enabled rule) choose a rule from the active component and all of them fire at once. Using the rules, the system passes from one configuration to another configuration; such a step is called a *transition*.

In a component- l -restricted transition, we say that the symbol 1 is generated if at least one rule with label l is used and a spike is sent out to the environment by the output neuron and the symbol 0 is generated if no spike is sent out to the environment. Similar to CD grammar systems, several cooperation strategies among the components can be considered: we here consider only the five basic ones.

For two configurations \mathcal{C} and \mathcal{C}' , we write $\mathcal{C} \Longrightarrow_l^* \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^{=k} \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^{\leq k} \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^{\geq k} \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^t \mathcal{C}'$, for some $k \geq 1$, if configuration \mathcal{C}' can be reached from \mathcal{C} as follows: (1) by any number of transitions, (2) by k transition steps, (3) by at most k transition steps, (4) by at least k transition steps, (5) by a sequence of transition steps using rules from l th component of each neuron, which cannot be continued, respectively.

A *computation* of Π is a finite or infinite sequence of transitions starting from the initial configuration, and every configuration appearing in such a sequence is called reachable. Therefore a finite (step) computation γ_α of Π in the mode $\alpha \in \{*, t\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$, is defined as $\gamma_\alpha = \mathcal{C}_0 \xrightarrow{j_1}^\alpha \mathcal{C}_1 \xrightarrow{j_2}^\alpha \dots \xrightarrow{j_y}^\alpha \mathcal{C}_y$ for some $y \geq 1$, $1 \leq j_y \leq p$, where \mathcal{C}_0 is the initial configuration. A computation γ_α of Π halts when the system reaches a configuration where no rule can be used as per the cooperating protocol α (i.e., the SN P system has halted). This paper only works in the terminating mode, so for convenience the mode is not explicitly used in the definitions hereafter.

With any computation, halting or not, we associate a spike train: a sequence of digits 0 and 1, with 1 appearing at positions corresponding to those steps when the output neuron sent a spike out of the system. With a spike train we can associate various numbers, which can be considered as generated by an SN P system. For instance, the distance in time between the first two spikes, between all consecutive spikes, the total number of spikes (in the case of halting computations), and so on.

It is clear that the standard SN P system introduced in [5] is a special case of the cooperating SN P system where the number of components is one. Similar to the standard SN P system, there are various ways of using this device. In the generative mode, one of the neurons is considered as the output neuron and the

spikes of the output neuron are sent to the environment. SN P systems can also be used for generating sets of vectors, by considering several output neurons, $\sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_v}$. In this case, the system is called a v -output SN P system. Here a vector of numbers, (n_1, n_2, \dots, n_v) , is said to be generated by the system if n_j is the number corresponding to the spike train from σ_{i_j} , where $1 \leq j \leq v$.

We denote by $C_p N_{gen}^{max}(II)$ [$C_p P S_{gen}^{max}(II)$] the set of numbers [of vectors, resp.] generated by a p -component SN P system II in a maximally parallel manner, and by $NC_p Spik_2 P_m^{max}(\beta)$ [$PsC_p Spik_2 P_m^{max}(\beta)$], $\beta \in \{gene, unb, boun\}$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by cooperating SN P systems of type β (*gene* stands for general, *unb* for unbounded, *boun* for bounded), with at most m neurons and p components. When m is not bounded, it is replaced by $*$. The subscript 2 reminds us of the fact that we count the number of steps elapsed between the first two spikes.

An SN P system can also work in the accepting mode: a neuron is designated as the input neuron and two spikes are introduced in it at an interval of n steps; input n is encoded by $2n$ in the input register; the number n is accepted if the computation halts.

In the asynchronous case, in each time unit, any neuron is free to use a rule or not. Even if enabled, a rule is not necessarily applied, the neuron can remain still in spite of the fact that it contains rules which are enabled by its contents. If the contents of the neuron are not changed, a rule which was enabled in a step t can fire later. If new spikes are received, then it is possible that other rules will be enabled – and applied or not. This way of using the rules also applies to the output neuron, hence now the distance in time between the spikes sent out by the system is no longer relevant. That is why, for asynchronous SN P systems we take as the result of a computation the total number of spikes sent out; this, in turn, makes necessary considering only halting computations (the computations never halting are ignored, they provide no output). We denote by $C_p N_{gen}^{nsyn}(II)$ [$C_p P S_{gen}^{nsyn}(II)$] the set of numbers [of vectors, resp.] generated by an asynchronous cooperating SN P system II with p components, and by $NC_p Spik_{tot} P_m^{nsyn}(\beta)$ [$PsC_p Spik_{tot} P_m^{nsyn}(\beta)$], $\beta \in \{gene, unb, boun\}$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by an asynchronous cooperating SN P systems of type β , with at most m neurons and p components. When m is not bounded, it is replaced by $*$. The subscript *tot* reminds us of the fact that we count all spikes sent to the environment.

In the strongly sequential case, in each neuron, in each time unit, at least one neuron contains a fireable rule and exactly one of them is chosen to fire non-deterministically. Here, the output can be interpreted in any of the earlier suggested ways. In this paper, we consider the distance in time between the first two spikes. We denote by $C_p N_{gen}^{sseq}(II)$ [$C_p P S_{gen}^{sseq}(II)$] the set of numbers [of vectors, resp.] generated by a strongly sequential cooperating SN P system II , and by $NC_p Spik_2 P_m^{sseq}(\beta)$ [$PsC_p Spik_2 P_m^{sseq}(\beta)$], $\beta \in \{gene, unb, boun\}$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by strongly sequential cooperating SN P systems of type β , with at most m neurons and p components. When m is not bounded, it is replaced by $*$.

4 Computational completeness of asynchronous SN P systems with two components using standard rules

We pass now to prove that the power of two components in standard neurons (where standard rules, producing one spike at a time, are used) can compensate for the loss of power entailed by removing the synchronization.

Theorem 1. $NC_2Spik_{tot}P_*^{nsyn}(gene) = NRE$.

Proof. We only have to prove the inclusion $NRE \subseteq NC_2Spik_{tot}P_*^{nsyn}(gene)$, and to this aim, we use the characterization of NRE by means of register machines used in the generating mode.

Let $M = (m, H, l_0, l_h, I)$ be a register machine with m registers, having the properties specified above: the result of a computation is the number stored in register 1 at the end of the computation and this register is never decremented during the computation.

What we want is an asynchronous SN P system with two components Π which (1) simulates the register machine M , and (2) has its output neuron emitting the number of spikes equal to the number computed by M .

Instead of specifying all technical details of the construction, we present the three main types of modules of the system Π , with the neurons, their rules, and their synapses represented graphically. In turn, simulating M means to simulate the *ADD* instructions and the *SUB* instructions. Thus, we will have one type of module associated with *ADD* instructions, one associated with *SUB* instructions, and one dealing with the spiking of the output neuron (a *FIN* module). The modules of the three types are given in Figs. 1, 2 and 3 respectively.

For each register r of M , we consider a neuron σ_r in Π whose contents corresponds to the contents of the register. Specifically, if the register r holds the number $n > 0$, then the neuron σ_r will contain $2n$ spikes. With each label l_i of an instruction in M , we also associate a neuron σ_{l_i} with a single rule $a \rightarrow a$ in its first component and no rules in its second component. There are also some auxiliary neurons $\sigma_{l_i, q}$, $q = 1, 2, 3, \dots$, thus precisely identified by label l_i . Initially, all these neurons are empty, with the exception of the neuron σ_{l_0} associated with the start label of M , which contains a single spike. This means that this neuron is activated. During the computation, the neuron σ_{l_i} which receives a spike will become active in its first component. Thus, simulating an instruction $l_i : (OP(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the register r as requested by *OP*, then introducing a spike in one of the neurons σ_{l_j} , σ_{l_k} which becomes active in this way. When activating the neuron σ_{l_h} , associated with the halting label of M , the computation in M is completely simulated in Π ; we will then send to the environment a number of spikes equal to the number stored in the register 1 of M . Neuron σ_1 is the output neuron of the system.

Simulating $l_i : (ADD(r), l_j, l_k)$ (module *ADD* in Fig. 1).

The initial instruction, that labelled l_0 , is an *ADD* instruction. Assume that we are in a step when we have to simulate an instruction $l_i : (ADD(r), l_j, l_k)$,

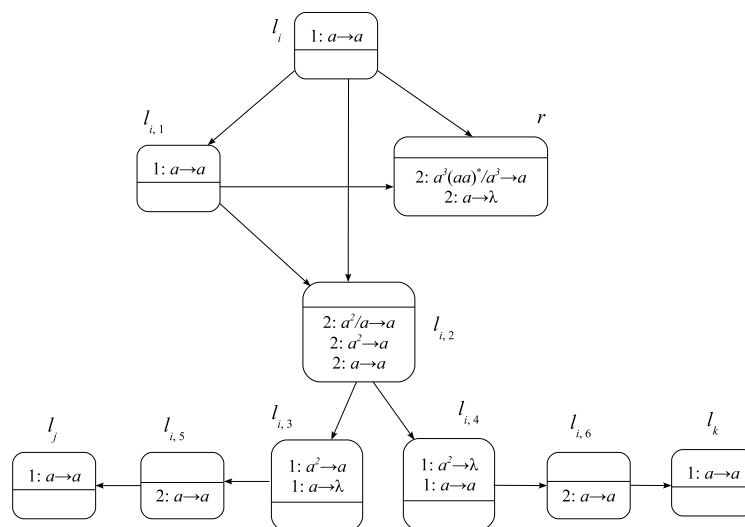


Fig. 1. ADD module: simulation of $l_i : (ADD(r), l_j, l_k)$

with a spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spikes in any other neurons, except in those associated with registers. Even if the system is in the second component at the time, it must switch over to the first component, since we are working in the terminating mode and there are no rules in the second component which are currently applicable anywhere in the system.

Having a spike inside and now in the first component, neuron σ_{l_i} can fire, and at some time it will do it, producing a spike. This spike will simultaneously go to neurons σ_r , $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$. The neurons σ_r and $\sigma_{l_{i,2}}$ cannot spike because the firing rules are present in their second components. The neuron $\sigma_{l_{i,1}}$ will spike at some time, then a spike will simultaneously go to the neurons σ_r and $\sigma_{l_{i,2}}$. Since no rules are enabled in the first component, the system switches to the second component. Before the system switches, the neuron σ_r receives two spikes from σ_{l_i} and $\sigma_{l_{i,1}}$, thus simulating the increase of the value of register r with 1. Now the system is in the second component. The neuron $\sigma_{l_{i,2}}$ has two spikes and it can fire by choosing one of its rules $a^2/a \rightarrow a$ or $a^2 \rightarrow a$ non-deterministically. If the neuron $\sigma_{l_{i,2}}$ uses its first rule $a^2/a \rightarrow a$, then it consumes one spike and sends a spike to each of the neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$. The neuron $\sigma_{l_{i,2}}$ is left with one spike and thus it has an enabled rule $a \rightarrow a$. The system switches to the first component only when no rules are enabled in the neuron $\sigma_{l_{i,2}}$. When the neuron $\sigma_{l_{i,2}}$ fires for the second time, neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ receive another spike and the system switches to the first component. The neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ have enabled rules and they can fire. The system will be in the first component as long as enabled rules are present in $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$. After some time, the neuron $\sigma_{l_{i,3}}$ uses its spiking rule and sends a spike to $\sigma_{l_{i,5}}$ and the neuron $\sigma_{l_{i,4}}$ forgets its

spikes. So eventually neuron $\sigma_{l_{i,5}}$ fires and sends a spike to σ_{l_j} , thus activating it.

If the neuron $\sigma_{l_{i,2}}$ uses its second rule $a^2 \rightarrow a$, then each of the neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ receive one spike finally. After some time, the neuron $\sigma_{l_{i,4}}$ uses its spiking rule and sends a spike to $\sigma_{l_{i,6}}$ and the neuron $\sigma_{l_{i,5}}$ forgets its spikes. So after some time, neuron $\sigma_{l_{i,6}}$ fires and sends a spike to σ_{l_k} , thus activating it. Therefore, from the firing of neuron σ_{l_i} , the system adds two spikes to neuron σ_r and non-deterministically fires one of the neurons σ_{l_j} and σ_{l_k} . Consequently, the simulation of the *ADD* instruction is possible in Π .

Simulating $l_i : (SUB(r), l_j, l_k)$ (module *SUB* in Fig. 2).

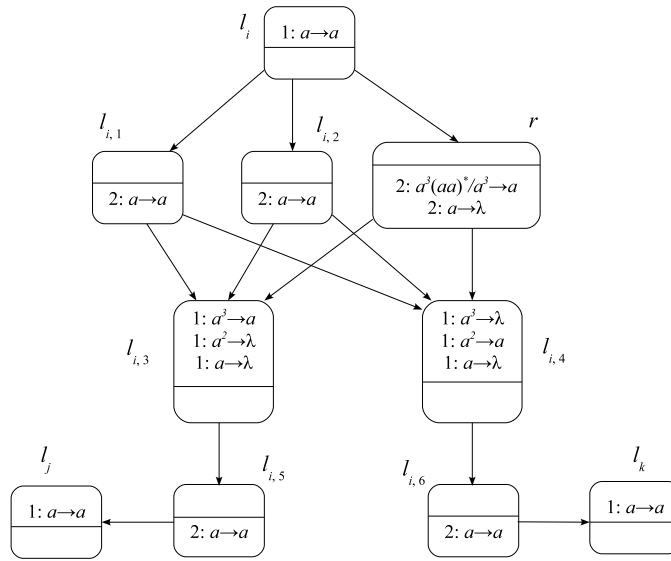


Fig. 2. SUB module: simulation of $l_i : (SUB(r), l_j, l_k)$

Let us now examine Fig. 2, starting from the situation of having a spike in neuron σ_{l_i} and no spike in other neurons, except neurons associated with registers; assume that neuron σ_r holds a number of spikes of the form $2n, n \geq 0$. Sometime, the neuron σ_{l_i} will fire and a spike goes immediately to each of the neurons $\sigma_{l_{i,1}}, \sigma_{l_{i,2}}$ and σ_r . The system must switch over to the second component, since we are working in the terminating mode and there are no rules in the first component which are currently applicable anywhere in the system.

If σ_r does not contain any spikes to begin with (this corresponds to the case when register r is empty), then eventually the spike sent by σ_{l_i} gets forgotten by virtue of the rule $a \rightarrow \lambda$ and σ_r is again left with no spikes, indicating that it is still zero. Eventually, neurons $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$ also send spikes using their rule $a \rightarrow a$. Thus, neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ end up with 2 spikes each and the system switches to the first component. After some steps, $\sigma_{l_{i,3}}$ forgets the two spikes

through the rule $a^2 \rightarrow \lambda$ and the neuron $\sigma_{l_{i,4}}$ fires using its rule $a^2 \rightarrow a$. With no rules applicable in the first component, the system switches to the second component and eventually neuron $\sigma_{l_{i,6}}$ sends a spike to neuron σ_{l_k} , as required, thus finishing the simulation of the *SUB* instruction for the case when register r is empty.

If neuron σ_r has $2n$ spikes to begin with, where $n \geq 1$, then after some steps, the rule $a(aa)^+/a^3 \rightarrow a$ is used in σ_r . Hence, σ_r now has two spikes less than what it began indicating that r has been reduced by 1. Further, neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ end up with 3 spikes each. After some steps, $\sigma_{l_{i,4}}$ forgets the three spikes through the rule $a^3 \rightarrow \lambda$ and the neuron $\sigma_{l_{i,3}}$ fires using its rule $a^3 \rightarrow a$. With no rules applicable in the first component, the system switches to the second component and eventually neuron $\sigma_{l_{i,5}}$ sends a spike to neuron σ_{l_j} , thus completing the simulation of the decrement case of the *SUB* instruction.

What remains to be examined is the possible interference between *SUB* modules. Note that there may be more than a single *SUB* instruction involving the same register r . Assume that we simulate the instruction $l_i : (SUB(r), l_j, l_k)$, hence neuron σ_r sends a spike to all neurons of the form $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$ for which there is an instruction $l_{i'} : (SUB(r), l_{j'}, l_{k'})$ in M . These spikes will be forgotten using the rule $a \rightarrow \lambda$ and this is the correct continuation of the computation. Note that the system will be in the first component as long as any spikes are present in the neurons of the form $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$. Thus, the neurons $\sigma_{l_{i,5}}$ and $\sigma_{l_{i,6}}$ will become active only after the forgetting rule $a \rightarrow \lambda$ is applied in each neuron of the form $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$.

This means that the simulation of the *SUB* instruction is correct, we started from l_i and ended in l_j if the register was non-empty (and we decreased it by one), and in l_k if the register was empty.

Simulating $l_h : (HALT)$ (module *FIN* in Fig. 3).

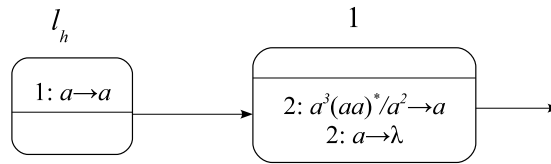


Fig. 3. *FIN* module: simulation of $l_h : HALT$

When the neuron σ_{l_h} is activated, it (eventually) sends one spike to neuron σ_1 , corresponding to the register 1 of M . From now on, this neuron can fire, and it sends out one spike for each two spikes present in it, hence the system will emit a number of spikes which corresponds to the contents of the register 1 of M at the end of the computation (after reaching the instruction $l_h : HALT$).

Consequently, $C_2 N_{gen}^{nsyn}(II) = N(M)$ and this completes the proof. □

Clearly, the previous construction is the same for the accepting mode, and can be carried out for deterministic register machines (the *ADD* instructions are of the form $l_i : (ADD(r), l_j)$). Similarly, if the result of a computation is defined as the number of spikes present in a specified neuron in the halting configuration, then the previous construction is the same, we only have to add one further neuron which is designated as the output neuron and which collects all spikes emitted by neuron σ_1 .

Theorem 1 can easily be extended by allowing more output neurons and then simulating a v -output register machine, producing in this way sets of vectors of natural numbers.

Theorem 2. $PsC_2Spik_{tot}P_*^{n, syn}(gene) = PsRE$.

5 Sequential spiking neural P systems with two components

In this section, we restrict the model to operate in a sequential manner. Before considering the power of sequential SN P systems with two components, we first recall some results from [4] on the power of the sequential SN P systems with one component.

1. Sequential SN P systems with general neurons are universal.
2. Sequential SN P systems with unbounded neurons are universal.
3. Strongly sequential SN P systems with general neurons are universal.
4. Strongly sequential SN P systems with unbounded neurons are not universal.

The paper [4] makes use of delayed rules to achieve synchronization. Here the synchronization can be achieved by switching between the components and hence delayed rules are not required. Here we prove that two component strongly sequential SN P systems with standard unbounded neurons without any delay are computationally complete.

Theorem 3. $NC_2Spik_2P_*^{sseq}(unb) = NRE$.

Proof. Given some register machine M generating a set $N(M)$, we can simulate M with a strongly sequential unbounded SN P Π having two components which generates the set $C_2N_{gen}^{sseq}(\Pi) = \{x \mid x \in N(M)\}$. The SN P Π 's initial configuration will again start with the initial configuration for each module along with a single spike in neuron σ_{l_0} .

To create a strongly sequential unbounded SN P generating exactly $N(M)$ use the same ideas and methods given in Theorem 1. The *ADD* module is the same as the one shown in Fig. 1 but we remove the rule $2 : a \rightarrow \lambda$ from the neuron σ_r so that the subsystem becomes unbounded. Since no rules from σ_r are fired in the *ADD* module, the subsystem works correctly even in the sequential mode.

The new subtraction module is shown in Fig. 4. It is initiated with a single spike in neuron σ_{l_i} which immediately sends a spike to neurons σ_r , $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$

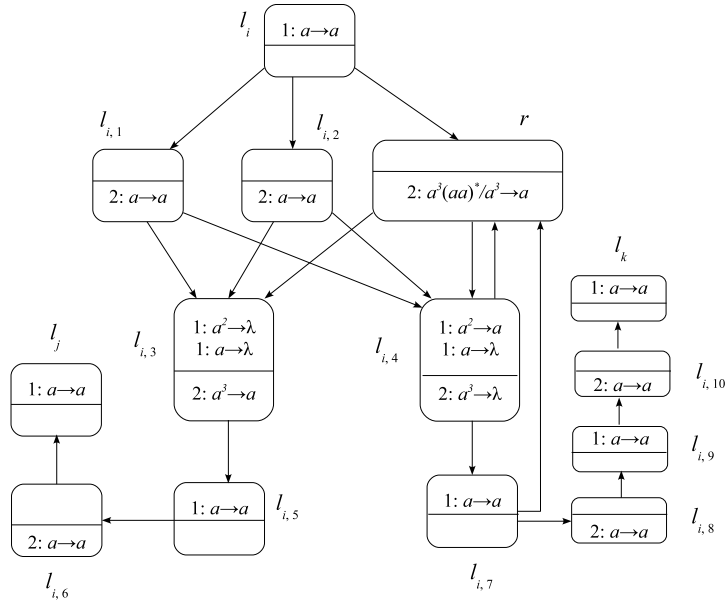


Fig. 4. Strongly sequential unbounded two component SN P *SUB* module

at time $t + 1$ (where t is the time the initial spike is sent to neuron σ_{l_i}). If the value in the register r is not zero then the three neurons non-deterministically spike during the next three steps (time $t + 2$, $t + 3$, and $t + 4$). This causes neuron $\sigma_{l_{i,3}}$ to spike and neuron $\sigma_{l_{i,4}}$ to forget sequentially during the following two time steps (time $t + 5$ and $t + 6$). Since no rules are enabled in the second component, the system switches to the first component. In the step $t + 7$, neuron $\sigma_{l_{i,5}}$ fires and sends a spike to $\sigma_{l_{i,6}}$. Since neuron σ_r sends spikes to all neurons $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$ where $l_{i'} : (SUB(r), l_{j'}, l_{k'})$, these neurons receive a single spike during the computation of instruction l_i . These spikes must be forgotten before the next instruction executes. Here, the system switches to the second component and fires the rule in the neuron $\sigma_{l_{i,6}}$ only after all spikes are removed from the neurons $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$ using their forgetting rule $a \rightarrow \lambda$ present in their first components. When the neuron $\sigma_{l_{i,6}}$ fires, it initiates the instruction module l_j .

If σ_r does not contain any spikes to begin with (this corresponds to the case when register r is empty), then the neuron σ_r does not fire and the neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ receive two spikes each. Since no rules are enabled in the second component, the system switches to the first component. This causes neurons $\sigma_{l_{i,3}}$ to forget and $\sigma_{l_{i,4}}$ to spike sequentially during the following two time steps (time $t + 4$ and $t + 5$). The spike from $\sigma_{l_{i,4}}$ goes simultaneously to neurons σ_r and $\sigma_{l_{i,7}}$ in time step $t + 6$. Neuron $\sigma_{l_{i,7}}$ sends a spike to σ_r and $\sigma_{l_{i,8}}$ in time step $t + 7$. Since no rules are enabled in the first component, the system switches to the second component. Now the neuron σ_r has three spikes. The neurons σ_r and

$\sigma_{l_{i,8}}$ fire sequentially in the next two steps $t + 8$ and $t + 9$. Thus the contents of σ_r is cleared indicating that r remains zero, as required. The spike from σ_r goes to $\sigma_{l_{i,3}}$, $\sigma_{l_{i,4}}$ and all neurons $\sigma_{l_{i',4}}$, where $l_{i'} : (SUB(r), l_{j'}, l_{k'})$. The neurons $\sigma_{l_{i,9}}$ and $\sigma_{l_{i,10}}$ with rules in different components ensures that the spikes in $\sigma_{l_{i,3}}$, $\sigma_{l_{i,4}}$ and all $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$ are forgotten before enabling the instruction module l_k as the spike received by $\sigma_{l_{i,8}}$ (from $\sigma_{l_{i,7}}$) percolates through $\sigma_{l_{i,9}}$ and $\sigma_{l_{i,10}}$ to l_k .

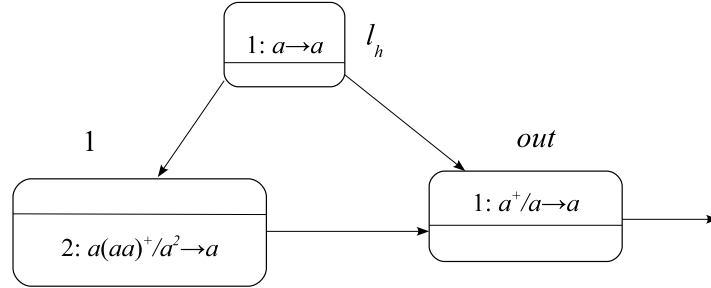


Fig. 5. Strongly sequential unbounded two component SN P output module

To simulate $l_h : (HALT)$, we create the module given in Fig. 5. When neuron l_h receives a spike, it fires and sends a spike to neurons σ_1 and σ_{out} with the system in the first component (it will switch to the first component even otherwise as only rules in the first component are enabled and we are working in the terminating mode). Let t be the moment when neuron l_h fires. Suppose the number stored in the register 1 of M is n .

At step $t + 1$, neuron σ_{out} fires for the first time sending its spike to the environment. The number of steps from this spike to the next one is the number computed by the system. Since no rule is enabled in the first component, the system switches to the second component. Now the neuron σ_1 spikes during the next n steps. The neuron σ_{out} will become active only after $2n$ spikes are removed from σ_1 . So at time $t + n + 1$, the system again switches to the first component and the neuron σ_{out} fires for the second time. The interval between the two spikes emitted by σ_{out} is $(t + n + 1) - (t + 1) = n$, which is the number stored in the register 1 of M . The system halts after $n - 1$ steps with all neurons empty except neuron σ_1 which contains a spike. \square

Theorem 3 can easily be extended by allowing more output neurons and then simulating a v -output register machine, producing in this way sets of vectors of natural numbers.

Theorem 4. $PsC_2Spik_2P_*^{seq}(unb) = PsRE$.

One more observation is that the module given in Fig. 4 works even if the system is asynchronous. It is now possible to construct a new system with *ADD* module shown in Fig. 1 without the rule $2 : a \rightarrow \lambda$ in the neuron σ_r , the *SUB*

module given in Fig. 4 and the *FIN* module given in Fig. 3 without the rule $2 : a \rightarrow \lambda$ in the neuron σ_1 which would be unbounded and work correctly in the case of an asynchronous system. Hence, we have the following two theorems.

Theorem 5. $NC_2Spik_{tot}P_*^{nsyn}(unb) = NRE$.

Theorem 6. $PsC_2Spik_{tot}P_*^{nsyn}(unb) = PsRE$.

Finally, the system constructed in Section 4 with the *FIN* module in Fig. 5 would work for sequential systems. Hence, we have the following two theorems.

Theorem 7. $NC_2Spik_2P_*^{sseq}(gene) = NRE$.

Theorem 8. $PsC_2Spik_2P_*^{sseq}(gene) = PsRE$.

6 Conclusion and discussion

The usual SN P systems operate in a maximally parallel manner. This model was shown to be computationally complete even with a variety of additional restrictions on the rule types [8, 11]. In this paper, we introduced a spiking neural P system with cooperating rules. Computational completeness has been proved for asynchronous as well as sequential cooperating SN P systems with two components using unbounded as well as general neurons working in the terminating mode. This suggests that cooperating SN P systems are indeed more powerful by offering seamless synchronization without the use of any delays. Further work would include the construction of small universal systems. It would also be interesting to consider the languages generated by these systems using different number of components. Further, this paper considers only the terminating mode, which is known to be more powerful than others in the case of CD grammar systems. A discussion on if the same result holds for cooperating SN P systems working in other models would be worthwhile.

Acknowledgements The work was supported by EU project Development of Research Capacities of the Silesian University in Opava (CZ.1.07/2.3.00/30.0007) and European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

1. Cavaliere, M., Ibarra, O.H., Păun, Gh., Egecioglu, Ö., Ionescu, M., Woodworth, S.: Asynchronous spiking neural P systems, *Theoretical Computer Science*, **410** (24-25), 2352–2364 (2009).
2. Csuhaĵ-Varjú, E., Dassow, J.: On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics (EIK)*, **26**, 49–63 (1990).
3. Csuhaĵ-Varjú, E., Dassow, J., Kelemen, J., Păun, Gh.: Grammar Systems. A Grammatical Approach to Distribution and Cooperation, Gordon and Breach, London, (1994).

4. Ibarra, O.H., Woodworth, S., Yu, F., Păun, A.: On spiking neural P systems and partially blind counter machines, *Natural Computing* **7**(1), 3–19 (2008).
5. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems, *Fundamenta Informaticae*, **71**, 279–308 (2006).
6. Meersman, R., Rozenberg, G.: Cooperating grammar systems, Proceedings of Mathematical Foundations of Computer Science, LNCS **64**, 364–374 (1978).
7. Minsky, M.: Computation – Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, NJ, (1967).
8. Pan, L., Păun, Gh.: Spiking Neural P Systems: An Improved Normal Form, *Theoretical Computer Science*, **411**, 906–918 (2010).
9. Păun, Gh., Rozenberg, G., Salomaa, A. (eds): Handbook of Membrane Computing, Oxford University Press, Oxford (2010).
10. Rozenberg, G., Salomaa, A. (eds): Handbook of Formal Languages. 3 volumes, Springer, Berlin, (1997).
11. Zeng, X., Zhang, X., Pan, L.: Homogeneous Spiking Neural P Systems, *Fundamenta Informaticae*, **97**, 1–20 (2009).

On the Dynamics of P Systems with Membrane Boundaries Extended Abstract

Tamás Mihálydeák¹ and Zoltán Ernő Csajbók²

¹ Department of Computer Science, Faculty of Informatics, University of Debrecen
Kassai út 26, H-4028 Debrecen, Hungary

`mihalydeak.tamas@inf.unideb.hu`

² Department of Health Informatics, Faculty of Health, University of Debrecen,
Sóstói út 2-4, H-4400 Nyíregyháza, Hungary

`csajbok.zoltan@foh.unideb.hu`

Abstract. There are many different variants of P systems which are used as a formalism to build more realistic models for complex biotic/chemical systems. In real biotic/chemical interactions, objects have to be close enough to membranes so that they are able to pass through them. Recently, the authors have proposed a new framework modeling the abstract notion of “closeness to membranes” in P systems by the help of multiset approximation spaces. The communication rules are restricted to these boundaries of membranes. This paper is addressed to reveal a possible intuitive meaning of this two-component structure.

Keywords: P systems, approximation spaces, potential energy, stable states

1 Introduction

The P system model was invented by Păun around the millennium [8–10]. In cell-like P systems, membranes delimit compartments (regions) arranged in a hierarchical structure. Each region is associated with a finite multiset of objects and a finite set of rules modeling reactions inside regions (evolution rules) and movements of objects through membranes (communication rules).

Communication rules are powerful means in membrane systems [7, 2, 11]. However, real biotic/chemical interactions represented by communication rules in P systems are taken place in the vicinity of membranes. Therefore, such an abstract concept of “closeness to membranes” is required which does not decrease the power of communication rules but controls their functioning in some ways.

Rough set theory, Pawlak’s classical theory of set approximations [5, 6] gives a plausible opportunity to model boundary zones around sets. We worked out its generalization for multisets called the multiset approximation space. This paper is addressed to show a possible biologically/chemically inspired intuitive meaning of the two-component structure, i.e., membrane system together with multiset approximation space.

2 P Systems with Membrane Boundaries

A P system with membrane boundaries which was first proposed in [3] has a two-component structure: 1) a communicating P system; and 2) a multiset approximation space. Using the multiset approximation technique, the abstract notion of closeness around membranes is specified. The communication rules are restricted to membrane boundaries. This approach is restrictive by nature but preserves the two important principles of P systems, the maximal parallelism and the nondeterminism. Hence, the membrane computations can be controlled to some extent [1, 4].

Among others, robustness, stability, equilibrium, periodicity are important aspects of real biotic/chemical processes. In the following, within the scope of the proposed framework, the question of stability in P systems will be investigated.

Formal definitions of all notions mentioned below can be found in [1, 3, 4].

A multiset approximation space, the one component of the proposed model, is defined over a finite alphabet U , and denoted by $MAS(U)$. It has four basic constituents:

- *Domain* — a set of finite multisets, msets for short, over U whose members are approximated. The msets associated with the regions belong to the domain.
- *Base system* — a set of some beforehand detached msets from the domain serving as the basis for the approximation process. Its members are the *base msets*.
- The set of *definable msets* deriving from base msets. They are possible approximations of the members of the domain. Base msets and the empty mset are always definable.
- *Approximation pair* — determining the lower and upper approximations of the msets of the domain as definable msets.

In chemical systems, energy is stored via chemical bonds (ionic or covalent) which establish a coherent unit from the atoms of elements, namely, a compound. Stored energy is called potential because it has the “potential” to do work. In Nature, the lower the potential energy of a system, the more stable it is. Moreover, natural systems, left to themselves, attempt to reach the configuration with the lowest energy possible under a given set of constraints.

In multiset approximation space $MAS(U)$, each base mset consists of objects which together form a coherent unit, too. Hence, base msets can be taken as the representation of compounds which store potential energy. Therefore, they represent the stable state of their constituent objects, and they have lower (potential) energy together as they would have separately.

The other component, the communication P system Π is also defined over a finite alphabet. In the framework, it is a cell-like system, i.e., its regions form a hierarchical structure. Sets of rules associated with the regions consist of solely communications rules of symport/antiport type. Having given the communication P system Π , an mset approximation space, denoted by $MAS(\Pi)$, can be created over the finite alphabet of the P system.

It is assumed that the msets associated with the regions belong to the domain of $MAS(I)$, and so their lower/upper approximations and boundaries can be formed in $MAS(I)$. However, these boundaries do not obey the membrane structure in general. Thus, they are adjusted to the membrane structure by a new mapping which gives the actual membrane boundaries.

Membrane boundaries are collections of base msets. A very important aspect of these base msets is that all of them are split into two parts by the membranes. They evidently form the inner and outer parts of the boundaries. In both parts, bisected base msets are not stable from the energetic point of view. In other words, they are in “excited states”, i.e., they have higher energy as they would have together as a coherent unit, namely, a base mset. As it was mentioned earlier, a natural system, left to itself, attempt to reach the configuration with the lowest energy if possible. Accordingly, base msets bisected by membranes are ready for moving towards the stable states of lower potential energy. However, they can reach these states only in the case if the adequate objects are able to pass through the membranes.

The movements of objects through membranes are regulated by communication rules restricted to membrane boundaries. Consequently, a base mset split into two parts can only put in a lower potential energy state, i.e., a more stable state if the communication rules make possible that the objects form a base mset again. If it happens, the (re)combined base mset wholly gets inside/outside the region. Then, at the same time, it is removed from the membrane boundary as well, i.e., the boundary changes.

Anyway, after each computation step in the framework, boundaries, both inner and outer parts, have to be recalculated. Then, the membrane computation can start again. It will finish if there is no applicable communication rule.

Acknowledgements

The publication was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund.

The authors are thankful to the anonymous referees for their constructive comments.

References

1. Csajbók, Z.E., Mihálydeák, T.: Maximal parallelism in membrane systems with generated membrane boundaries. In: Beckmann, A., Csuhaj-Varjú, E., Meer, K. (eds.) *Language, Life, Limits*. 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23-27, 2014. Proceedings. LNCS, vol. 8493, pp. 103–112. Springer International Publishing, Switzerland (2014)
2. Freund, R., Păun, A.: Membrane systems with symport/antiport rules: Universality results. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) *Membrane Computing*, LNCS, vol. 2597, pp. 270–287. Springer Berlin Heidelberg (2003)

3. Mihálydeák, T., Csajbók, Z.E.: Membranes with boundaries. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, Gy. (eds.) *Membrane Computing. 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*. LNCS, vol. 7762, pp. 277–294. Springer-Verlag, Berlin Heidelberg (2013)
4. Mihálydeák, T., Csajbók, Z.E., Takács, P.: Communication rules controlled by generated membrane boundaries. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Salomaa, A. (eds.) *Membrane Computing, 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*. LNCS, vol. 8340, pp. 265–279. Springer, Berlin Heidelberg (2014)
5. Pawlak, Z.: Rough sets. *International Journal of Computer and Information Sciences* 11(5), 341–356 (1982)
6. Pawlak, Z.: *Rough Sets: Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, Dordrecht (1991)
7. Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. *New Generation Computing* 20(3), 295–305 (September 2002)
8. Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences* 61(1), 108–143 (2000)
9. Păun, Gh.: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin (2002)
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford Handbooks, Oxford University Press, Inc., New York, NY, USA (2010)
11. Sosík, P.: P systems versus register machines: Two universality proofs. In: Păun, Gh., Zandron, C. (eds.) *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Argeş, Romania, pp. 371–382 (2002)*

Parallel Thinning with Complex Objects and Actors

Radu Nicolescu

Department of Computer Science, The University of Auckland
Auckland, New Zealand, r.nicolescu@auckland.ac.nz

9 May, 2014

Abstract

Based on our earlier complex objects proposal, we present three novel concurrent membrane computing models for a fundamental image processing task: the thinning (or skeletonisation) of binary images, based on the classical Guo-Hall algorithm (A2). The first model is synchronous and uses one cell per pixel and relies on inter-cell parallelism; the second model is an asynchronous version of the first; the third model uses one single cell, with one sub-cellular object per pixel, and relies on intra-cell parallelism. The static and dynamic qualities of our models validate our complex objects proposal: (i) the proposed models are crisp (comparable to the best pseudocode); and (ii) complex objects concurrency and messaging can be efficiently emulated on a message-based Actors framework (which opens a novel research path).

Keywords: Membrane computing, P systems, inter-cell parallelism, intra-cell parallelism, Prolog terms, complex objects, generic rules, image processing, Guo-Hall algorithm, parallel and concurrent models, synchronous and asynchronous models, termination detection, message-based, Actor model.

1 Introduction

We have previously used complex objects to successfully model problems in a wide variety of domains: computer vision [8, 9, 14]; graph theory [18, 6]; distributed algorithms [2, 4, 5, 17, 25, 19]; high-level P systems programming [16, 15]; numerical P systems [16, 15]; NP-complete problems [16, 15, 19].

In this paper, we choose another test case, a fundamental image processing task: thinning (or skeletonisation) of black-and-white images. Specifically, we model Guo and Hall's algorithm A2 [10] and we assess the merits of our complex framework. We provide three closely related models, with a straightforward translation between them:

- A *synchronous multi-cell* model, based on a 1 : 1 mapping between pixels and cells and essentially using the *inter-cell parallelism*;
- An *asynchronous multi-cell* model, based on a 1 : 1 mapping between pixels and cells and essentially using the *inter-cell parallelism*; and

- A (synchronous) *single-cell* model, based on a 1 : 1 mapping between pixels and sub-cellular complex objects and essentially using the *intra-cell parallelism*.

Further, although not detailed here (for lack of space), we provide a direct emulation of our three models in the *asynchronous* Actors framework [11, 1], using F#'s mailbox processor library [24]. We conjecture that, given a good support for pattern matching, the translation from complex objects to Actors be largely automatised, not only for similar image processing tasks, but for many other applications (such as we earlier studied), for both synchronous and asynchronous cases.

We are aware of only a few other quality studies proposing membrane computing models for thinning algorithms or other real-life complex image processing tasks. Reina-Molina et al. [23] propose a traditional tissue system, without mentioning any experimental mapping, and raise an *open question*: if other membrane computing versions can provide “better” (in a very loose sense) models. Reina-Molina and Díaz-Pernil [22] discuss a similar tissue based system. Peña-Cantillana et al. [20] discuss a related cellular model, mapped on the CUDA platform. Díaz-Pernil et al. [3] propose a spiking neural model also mapped on the CUDA platform.

We think that our study is a partial answer to the above open question: our complex objects seem to support crisper models, with fixed-sized rulesets, independent on the size of the image. Moreover, our approach seems to be the first offering multiple but closely related solutions for an image processing task, such as a pair of “twin” multi-cell/single-cell solutions, or a pair of “twin” synchronous/asynchronous solutions. Also, we are not aware of any other attempt that map membrane computing to an Actors based platform, so this is another novel path explored in this paper.

This study leads us to formulate a couple of open questions (not followed here), about a distributed termination detection of this algorithm or the merits of a partial asynchronous version of this algorithm. We are not aware of many membrane computing studies that could be related to these open problems. For example, Nicolescu and Wu [19] and Wu [26] have recently studied distributed termination detection for diffusing (single source) algorithms, but probably more studies are needed on non-diffusing (multiple source) algorithms such as this (especially for scenarios when a master control node cannot be easily incorporated).

Because of space constraints, for the rest of the paper, we assume that the reader is already familiar with:

- basic definitions used in traditional *tissue-like transition P systems*, including state based rules, weak priority, promoters and inhibitors, e.g. as discussed in the membrane computing handbook [21];
- basic concepts of image processing and an understanding of Guo and Hall's *image thinning* algorithm A2 [10, 20];
- basic concepts of *concurrent processing with functional programming*, using the message-based *Actor* model, e.g. as described in Syme's monograph [24].

However, to ensure some degree of self-containment, following Nicolescu et al. recent papers [16, 15, 19], we discuss two important extensions used: *complex objects* and *generic rules* (if our complex objects presentation is not enough, then we also suggest a quick revision of basics of Prolog terms unification). Here we also extend the previously published format of complex objects and generic rules, by adding an optional *path* notation which enables efficient “micro-surgeries” on inner nested objects (without affecting their enclosing outer objects).

2 Background: Image Thinning

Thinning is the transformation of a digital binary image into a simplified, but topologically equivalent image, often called “skeleton”. The image pixels are divided into two categories: (i) foreground, conventionally *black* or **true** (i.e. ordinal 1); and (ii) background, conventionally *white* or **false** (i.e. ordinal 0). Typically, the thinning process repeatedly turns *black* pixels to *white* (conceptually “deleting” them from the foreground), while trying (a) to keep the topological connectivity, and (b) to balance the whitening around the so call medial lines. The algorithm stops when no further changes can be made.

Intuitively, Figure 1 shows a thinning example: (1a) the original image, and (1b) a possible skeleton. For efficiency, many thinning algorithms make such changes solely based on the current pixel’s 8-neighbourhood, where center pixel p_0 has the following neighbours (see also Figure 2a): p_1 at NW, p_2 at N, p_3 at NE, p_4 at E, p_5 at SE, p_6 at S, p_7 at SW, p_8 at W.

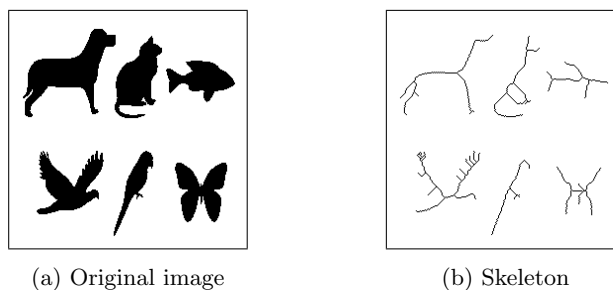


Figure 1: Sample thinning (with Guo and Hall A2)

Because of paper size limits, we only outline the critical steps of Guo and Hall’s algorithm A2 [10]; please see the original paper for more details on thinning and on this algorithm, including arguments for its correctness and efficiency.

Briefly, one partitions the pixels into an even/odd checkerboard pattern, where a pixel at (i, j) is even if $(i+j)\%2 = 0$, and odd otherwise. This algorithm makes alternative iterations, where at one step only even (or odd) black pixels are active and examined for possible changes. Figure 2b shows the essential pixel state changes of this algorithm: a white pixel remains white; a black pixel continuously alternates between an active and inactive state (determined by the checkerboard pattern and iteration number); and an active black pixel may turn white, under the specific conditions detailed below.

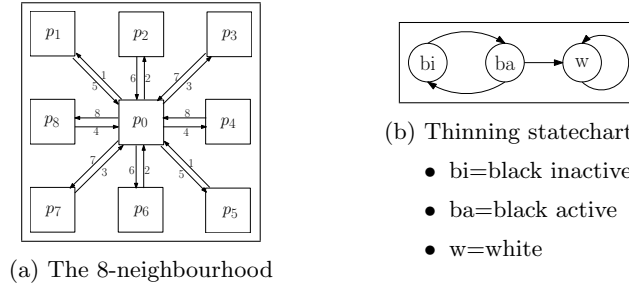


Figure 2: The “standard” 8-neighbourhood and algorithm statechart

Besides being in the active state, the following three conditions¹ must be simultaneously true for properly changing the colour of a *black* pixel p_0 (“deleting” it from the foreground):

- $\beta = 1$, where

$$\beta = \text{ord}(\neg((p_1 \wedge p_3 \wedge p_5 \wedge p_7) \vee (p_2 \wedge p_4 \wedge p_6 \wedge p_8)))$$

- $\gamma = 1$, where

$$\gamma = (\text{ord}((\neg p_2) \wedge (p_3 \vee p_4))) + (\text{ord}((\neg p_4) \wedge (p_5 \vee p_6))) + (\text{ord}((\neg p_6) \wedge (p_7 \vee p_8))) + (\text{ord}((\neg p_8) \wedge (p_1 \vee p_2)))$$

- $\delta > 1$, where

$$\delta = (\text{ord } p_1) + (\text{ord } p_2) + (\text{ord } p_3) + (\text{ord } p_4) + (\text{ord } p_5) + (\text{ord } p_6) + (\text{ord } p_7) + (\text{ord } p_8)$$

These three values can be each time computed from scratch, according to the formulas, or precomputed and stored in three fast access tables, each one storing all values for all $2^8 = 256$ combinations of p_1, p_2, \dots, p_8 . Or, even better, we can precompute all answers for the combined three conditions:

$$\psi = \text{ord}((\beta = 1) \wedge (\gamma = 1) \wedge (\delta > 1)).$$

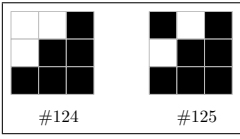
Also, to avoid special cases for pixels lying on the image borders, in many cases we can assume that all border pixels are background (white). In this paper, we follow these two optimisations.

Table 3a shows (in a compact way) a few lines of the β , γ , δ and ψ tables; the configurations of lines 124 and 125 are displayed in Figure 3b; the original article [10] explains the rationale of these three values (called 4 – *connection*, $C(P)$, and $B(P)$, respectively).

¹Function $\text{ord}(b)$ returns the ordinal value of boolean b , i.e. $\text{ord}(\perp) = 0, \text{ord}(\top) = 1$.

	$p_8, p_7, p_6, p_5, p_4, p_3, p_2, p_1$	β	γ	δ	ψ
0	0, 0, 0, 0, 0, 0, 0, 0	1	0	0	0
...
124	0, 1, 1, 1, 1, 1, 0, 0	1	1	5	1
125	0, 1, 1, 1, 1, 1, 0, 1	0	2	6	0
...
255	1, 1, 1, 1, 1, 1, 1, 1	0	0	8	0

(a) Excerpts of the β , γ , δ and ψ tables



(b) Configurations of lines 124 (centre pixel can be “deleted”) and 125 (centre pixel cannot be “deleted”)

Figure 3: Precomputed tables

This algorithm can be parallelised by allocating one distinct task for each pixel, provided that we have implicit or explicit *synchronisation barriers*, which ensure that the colour changes do not affect other pixels’ evaluation of the “deletion” condition. This condition is implicit in *synchronous* settings; however, in *asynchronous* settings (as typical in parallel multi-cores applications), we need two explicit synchronisation barriers. Listing 1 shows a parallel pseudocode of the essential steps of Guo and Hall’s algorithm A2 (using indentation to delimit blocks), assuming that colours are directly changed in the original pixels. We can safely use this approach, in a synchronous P system model, where all steps are implicitly synchronised.

Practically, to avoid excessive parallelisation overhead, one should reduce the number of tasks, by adequately partitioning the image and allocate a task to each subrange of pixels. Thus, the first line of Listing 1 should be replaced by the lines shown in Listing 2. This mixed approach is not further developed here, but is briefly mentioned again, as an open problem, in Section 5.

Listing 1: Synchronous parallel pseudocode of Guo and Hall’s algorithm A2

```

parallel for each pixel  $\pi[i, j]$  in image do
  if ( $\pi[i, j].\text{colour} = \text{black}$ ) then
     $\pi[i, j].\text{active} := ((i + j) \% 2 = 0)$ 
    while ( $\neg$  terminated) do // question: how do black cells know this?
       $\pi[i, j].\text{active} := \neg \pi[i, j].\text{active}$ 
      if ( $\pi[i, j].\text{active}$ ) then
        barrier // phase synchronisation
        local  $\psi := \dots$  // needs the colours of its 8 neighbours
        barrier // phase synchronisation
        if ( $\psi = 1$ ) then // i.e.  $(\beta = 1) \wedge (\gamma = 1) \wedge (\delta > 1)$ 
           $\pi[i, j].\text{colour} := \text{white}$ 
          break while
      else
        barrier // phase synchronisation
        idle
        barrier // phase synchronisation

```

This algorithm raises an interesting *termination* issue, in the case of a dis-

Listing 2: Partitioning pseudocode for Guo and Hall’s algorithm A2

```

 $\Pi$  = partition of input image // question: what is the best partition?
parallel for each subrange in  $\Pi$  do
  coroutine for each pixel  $\pi[i, j]$  in subrange do
    ... // the rest is identical

```

tributed implementation over a cluster of computing nodes: how do all nodes learn that the algorithm has terminated, promptly and efficiently? Is there a master node that supervises the whole process, perhaps centralising the number of deleted pixels at each step (as we assume here), or do the nodes concurrently run an ad-hoc termination detection control layer?

Another interesting question is to find well-behaved *asynchronous* versions of this algorithm. While a full asynchronous version could arguably return bad results (e.g. far from the medial line), a restricted, globally asynchronous but locally synchronous (GALS) version may still give exact results. In Section 6, we propose an asynchronous Actors based version, where we precisely know the number of expected messages at each phase, which enables an ad-hoc local synchronisation, without the overhead of more general synchroniser [12].

3 Background: Membrane Computing with Complex Objects

3.1 Complex objects

Complex objects play the roles of cellular micro-compartments or substructures, such as organelles, vesicles or cytoophidium assemblies (“snakes”), which are embedded in cells or travel between cells, but without having the full processing power of a complete cell. In our proposal, complex objects represent structured data that have no own processing power: they are acted upon by the rules of their enclosing cells.

Technically, our *complex objects*, are Prolog-like *first-order terms*, recursively built from *multisets* of atoms and variables. *Atoms* are typically denoted by lower case letters, such as a, b, c . *Variables* are typically denoted by uppercase letters, such as X, Y, Z . For improved readability, we also consider *anonymous variables*, which are denoted by underscores (“_”). Each underscore occurrence represents a *new* unnamed variable and indicates that something, in which we are not interested, must fill that slot.

Terms are either (i) simple atoms, or (ii) atoms (here called *functors*), followed by one or more parenthesized lists of comma-separated “arguments”, which are *multisets/bags* of other objects (terms or variables). Functors that are followed by more than one list of arguments are called *curried* (by analogy to functional programming) and, as we see later, are useful to precisely described deep ‘micro-surgical’ changes which only affect inner nested objects, without directly touching their enclosing outer objects. Terms that do *not* contain variables are called *ground*, e.g.:

- Ground terms: a , $a(b)$, $a(\lambda)$, $a(b, c)$, $a(b, \lambda)$, $a(\lambda, c)$, $a(\lambda, \lambda)$, $a(b(c))$, $a(bc)$, $a(bc(\lambda))$, $a(b(c)d(e))$, $a(b(c), d(e))$, $a(b(c), d(e(\lambda)))$, $a(bc^2, d, eg)$; or, a curried form: $a(bc^2)(d, eg)$.
- Terms which are not ground: $a(b, X)$, $a(b(X))$, $a(Xc)$, $a(bY)$, $a(XY)$, $a(XX)$, $a(XdY)$, $a(Xc())$, $a(b(X)d(e))$, $a(b(c), d(Y))$, $a(b(X), d(e(Y)))$, $a(b(X), d(e(X)))$; or, a curried form: $a(b(c))(d(Y))$; also, using anonymous variables: $a(b_)$, $a(X_)$, $a(b(X), d(e(-)))$.

Note that we may abbreviate the expression of complex objects by removing inner λ 's as explicit references to the empty bag, e.g. $a(\lambda) = a()$, $a(b, \lambda) = a(b,)$. Complex objects can be formally defined by the following grammar:

```

<term> ::= <atom> | <functor> ( '(' <arguments> ') ' )+
<functor> ::= <atom>
<arguments> ::= <bag-argument> ( ', ' <bag-argument> )*
<bag-argument> ::=  $\lambda$  | ( <term-or-var> )+
<term-or-var> ::= <term> | <variable>

```

Natural numbers. Natural numbers can be represented via *bags* containing repeated occurrences of the *same* atom. For example, considering that l represents the unary digit, then the following complex objects can be used to describe the contents of a virtual integer *variable* a : $a()$ — the value of a is 0; $a(l^3)$ — the value of a is 3. Nicolescu et al. [16, 15, 19] show how arithmetic operations can be efficiently modelled by P systems with complex objects.

Indexed symbols. Complex objects can sometimes be represented as *indexed symbols*, where lower-case indices stand for arguments, e.g. $a_k = a(k)$; this is especially convenient when indices represent numbers or cell IDs (which typically are numbers).

Unification. All terms (ground or not) can be (asymmetrically) *matched* against *ground* terms, using an ad-hoc version of *pattern matching*, more precisely, a *one-way first-order syntactic unification*, where an atom can only match another copy of itself, and a variable can match any bag of ground terms (including the empty bag, λ). This may create a combinatorial *non-determinism*, when a combination of two or more variables are matched against the same bag, in which case an arbitrary matching is chosen. For example:

- Matching $a(X, eY) = a(b(c), def)$ deterministically creates a single set of unifiers: $X, Y = b(c), df$.
- Matching $a(XY) = a(df)$ non-deterministically creates one of the following four sets of unifiers: $X, Y = \lambda, df$; $X, Y = df, \lambda$; $X, Y = d, f$; $X, Y = f, d$.
- However, matching $a(XY, Y) = a(def, e)$ deterministically creates a single set of unifiers: $X, Y = df, e$.

Performance note. If the rules avoid any matching non-determinism, then this proposal should not affect the performance of P simulators running on existing machines. Assuming that bags are already taken care of, e.g. via hash-tables, our proposed unification probably adds an almost linear factor. Let us

recall that, in similar contexts (no occurs check needed), Prolog unification algorithms can run in $O(ng(n))$ steps, where g is the inverse Ackermann function. Our conjecture must be proven though, as the novel presence of multisets may affect the performance.

Alternative notations. Using Lisp terms instead Prolog terms, we could use an equivalent notation, where the functor becomes the first term in a parent-sized expression, instead of preceding it. For example, the Prolog-like term $a(bc^2, d, eg)$ could be rewritten as the Lisp-like term, (a, bc^2, d, eg) , with additional commas to clearly separate items.

If there is no confusion with atomic symbols, simple complex terms with just one level of nesting can also be abbreviated by an indexed notation. For example, the term $a(i, j)$ could be rewritten as $a_{i,j}$.

3.2 Generic rules

By default, rules are applied top-down, in the so-called *weak priority* order. *Rules* may contain *any* kind of terms, ground and not-ground; however, in this proposal, *cells* can only contain *ground* terms.

Pattern matching. Rules are matched against cell contents using the above discussed *pattern matching*, which involves the rule's left-hand side, promoters and inhibitors. Moreover, the matching is *valid* only if, after substituting variables by their values, the rule's right-hand side contains ground terms only (so *no* free variables are injected in the cell or sent to its neighbours), as illustrated by the following sample scenario:

- The cell's *current content* includes the *ground term*:
 $n(l^{10}, n(l^{20}, f(l^{30}), f(l^{40})), f(l^{50}))$
- The following *rewriting rule* is considered:
 $n(X, n(Y, Y_1, Y_2), f(Z)) \rightarrow v(X) n(Y, Y_1, Y_2) v(Z)$
- Our pattern matching determines the following *unifiers*:
 $X = l^{10}, Y = l^{20}, Y_1 = l^{30}, Y_2 = l^{40}, Z = l^{50}$.
- This is a *valid* matching and, after *substitutions*, the rule's *right-hand* side gives the *new content*:
 $v(l^{10}) n(l^{20}, f(l^{30}), f(l^{40})) v(l^{50})$

Generic rules format. More generally, we consider rules of the following *generic* format (we call this format generic, because it actually defines templates involving variables):

$\begin{aligned} \text{current-state objects} \dots &\rightarrow_{\alpha} \text{target-state } [\text{immediate-objects}] \dots \\ &(\text{in-objects}) \dots (\text{out-objects})_{\delta} \dots \\ & \text{promoters} \dots \neg \text{inhibitors} \dots \end{aligned}$
--

Where:

- *States* are complex objects (which can be *matched*, as previously described).

- All *objects*, *promoters* and *inhibitors* are *bags of terms*, possibly containing *variables* (which are *matched* as previously described).
- *Out-objects* are sent, at the end of the step, to the cell's structural neighbours. These objects are enclosed in round parentheses which further indicate their destinations, above abbreviated as δ ; the most usual scenarios include: $(a) \downarrow_i$ indicates that a is sent to child i (unicast), $(a) \uparrow_i$ indicates that a is sent to parent i (unicast), $(a) \downarrow_{\forall}$ indicates that a is sent to all children (broadcast), $(a) \uparrow_{\forall}$ indicates that a is sent to all parents (broadcast), $(a) \updownarrow_{\forall}$ indicates that a is sent to all neighbours (broadcast).
- Both *immediate-objects* and *in-objects* remain in the current cell, but there is a subtle difference:
 - *in-objects* become available at the end of the current step only, as in traditional P systems (we can imagine that these are sent via an ad-hoc *loopback* channel);
 - *immediate-objects* become immediately available to (i) to the current rule, if it uses the **max** instantiation mode, and (ii) the succeeding rules (in weak priority order).

Immediate objects can substantially improve the runtime performance, which could be required for two main reasons: (i) to achieve parity with best traditional algorithms, and (ii) to ensure correctness when proper timing is logically critical. However, they are not used in the systems presented in this paper.

- Symbol $\alpha \in \{\min, \max\} \times \{\min, \max\}$, indicates a combined instantiation and rewriting mode, as further discussed below.

We often abbreviate by omitting the round parentheses that enclose in-objects. In other words, the traditional in-objects are the default. This respects the tradition and is especially useful for rulesets which do not use immediate objects.

Example. To explain our combined instantiation and rewriting mode, let us consider a cell, σ , containing three counter-like complex objects, $c(c(a))$, $c(c(a))$, $c(c(c(a)))$, and the four possible instantiation \otimes rewriting modes of the following “decrementing” rule:

$$(\rho_{\alpha}) S_1 c(c(X)) \rightarrow_{\alpha} S_2 c(X), \text{ where } \alpha \in \{\min, \max\} \times \{\min, \max\}.$$

1. If $\alpha = \min \otimes \min$, rule $\rho_{\min \otimes \min}$ nondeterministically generates and applies (in the **min** mode) *one* of the following two rule instances:

$$\begin{aligned} (\rho'_1) \quad S_1 c(c(a)) &\rightarrow_{\min} S_2 c(a) \quad \text{or} \\ (\rho''_1) \quad S_1 c(c(c(a))) &\rightarrow_{\min} S_2 c(c(a)). \end{aligned}$$

Using (ρ'_1) , cell σ ends with counters $c(a)$, $c(c(a))$, $c(c(c(a)))$. Using (ρ''_1) , cell σ ends with counters $c(c(a))$, $c(c(a))$, $c(c(a))$.

2. If $\alpha = \max \otimes \min$, rule $\rho_{\max \otimes \min}$ first generates and then applies (in the \min mode) the following *two* rule instances:

$$\begin{aligned} (\rho'_2) \quad S_1 c(c(a)) &\rightarrow_{\min} S_2 c(a) \quad \text{and} \\ (\rho''_2) \quad S_1 c(c(c(a))) &\rightarrow_{\min} S_2 c(c(a)). \end{aligned}$$

Using (ρ'_2) and (ρ''_2) , cell σ ends with counters $c(a)$, $c(c(a))$, $c(c(a))$.

3. If $\alpha = \min \otimes \max$, rule $\rho_{\min \otimes \max}$ nondeterministically generates and applies (in the \max mode) *one* of the following rule instances:

$$\begin{aligned} (\rho'_3) \quad S_1 c(c(a)) &\rightarrow_{\max} S_2 c(a) \quad \text{or} \\ (\rho''_3) \quad S_1 c(c(c(a))) &\rightarrow_{\max} S_2 c(c(a)). \end{aligned}$$

Using (ρ'_3) , cell σ ends with counters $c(a)$, $c(a)$, $c(c(c(a)))$. Using (ρ''_3) , cell σ ends with counters $c(c(a))$, $c(c(a))$, $c(c(a))$.

4. If $\alpha = \max \otimes \max$, rule $\rho_{\min \otimes \max}$ first generates and then applies (in the \max mode) the following *two* rule instances:

$$\begin{aligned} (\rho'_4) \quad S_1 c(c(a)) &\rightarrow_{\max} S_2 c(a) \quad \text{and} \\ (\rho''_4) \quad S_1 c(c(c(a))) &\rightarrow_{\max} S_2 c(c(a)). \end{aligned}$$

Using (ρ'_4) and (ρ''_4) , cell σ ends with counters $c(a)$, $c(a)$, $c(c(a))$.

The interpretation of $\min \otimes \min$, $\min \otimes \max$ and $\max \otimes \max$ modes is straightforward. While other interpretations could be considered, the mode $\max \otimes \min$ indicates that the generic rule is instantiated as *many* times as possible, without *superfluous* instances (i.e. without duplicates or instances which are not applicable) and each one of the instantiated rules is applied *once*, if possible.

If a rule does not contain any non-ground term, then it has only one possible instantiation: itself. Thus, in this case, the instantiation is an *idempotent* transformation, and the modes $\min \otimes \min$, $\min \otimes \max$, $\max \otimes \min$, $\max \otimes \max$ fall back onto traditional modes \min , \max , \min , \max , respectively.

Special cases. Simple scenarios involving generic rules are sometimes semantically equivalent to loop-based sets of non-generic rules. For example, consider the rule

$$S_1 a(I, J) \rightarrow_{\max \otimes \min} S_2 b(I) c(J),$$

where I and J are guaranteed to only match integers in ranges $[1, n]$ and $[1, m]$, respectively. Under these assumptions, this rule is equivalent to the following set of non-generic rules:

$$S_1 a(i, j) \rightarrow_{\min} S_2 b(i) c(j), \quad \forall i \in [1, n], j \in [1, m].$$

However, unification is a much more powerful concept, which cannot be generally reduced to simple loops.

Micro-surgery: operations that only affect inner nested objects. Such operations improve both the crispness and the efficiency of the rules. Consider

a cell that contains objects $o(abpq), r$ and a naive rule which attempts to change the inner b to a d , if an inner p and a top-level r are also present:

$$S_1 o(bR) \rightarrow_{\min \otimes \min} S_2 o(dR) \mid o(p-) r.$$

Unless we change the “standard” application rules, this rule fails, because an object locked as a promoter cannot be changed at the same time. We solve this problem without changing the standard application rules, by adding an access path to the inner objects needed. The *access path* is a slash delimited list of outer objects, in nesting order, which opens an inner bag for usual rewriting operations; these outer objects on the path are not themselves touched. For example, this modified rule will solve the problem:

$$S_1 o/b \rightarrow_{\min \otimes \min} S_2 o/d \mid o/p r.$$

This extension helps even more when we want to localise the changes to inner objects of a specific outer object. For example, consider a similar operation that needs to be applied on the innermost contents of object $o(i, j)(abpq)$, identified by its coordinates i, j .

$$S_1 o(i, j)/b \rightarrow_{\min \otimes \min} S_2 o(i, j)/d \mid o(i, j)/p r.$$

If all or most objects involved share the same path, than the path could qualify the whole rule; existing top-level objects could be qualified by usual path conventions, e.g. in our case, r could be explicitly qualified by either of $/$ or $../$:

$$o(i, j) :: S_1 b \rightarrow_{\min \otimes \min} S_2 d \mid p ../r.$$

Note that the usual rulesets are just a special case of this extension, when all rules are by default qualified with the root path $/$.

Note. For all modes, the instantiations are *conceptually* created when rules are tested for applicability and are also *ephemeral*, i.e. they disappear at the end of the step. P system implementations are encouraged to directly apply high-level generic rules, if this is more efficient (it usually is); they may, but need not, start by transforming high-level rules into low-level rules, by way of instantiations.

Benefits. This type of generic rules allow (i) a reasonably fast parsing and processing of subcomponents, and (ii) algorithm descriptions with *fixed size alphabets* and *fixed sized rulesets*, independent of the size of the problem and number of cells in the system (often impossible with only atomic symbols).

Synchronous vs asynchronous. In our models, we do not make any syntactic difference between the synchronous and asynchronous scenarios; this is strictly a *runtime* assumption [13]. Any model is able to run in both the synchronous and asynchronous runtime “engines”, albeit the results may differ.

As in traditional P systems, in the *synchronous* scenario, all rules in a step take exactly *one* time unit and then all message exchanges (including loopback messages for in-objects) are performed at the end of the step, in *zero* time (i.e. instantaneously). Alternatively, but logically equivalent, we can consider that rules in a step are performed in *zero* time (i.e. instantaneously) and then all message exchanges are performed in exactly *one* time unit. The second interpretation is useful, because it allows us to interpret synchronous runs as special cases of asynchronous runs.

In the *asynchronous* scenario, we still consider that rules in a step are performed in *zero* time (i.e. instantaneously), but then each message may take *any* finite real time to arrive at the destination. Additionally, unless otherwise specified, we also assume that messages traveling on the same directed arc follow a *FIFO* rule, i.e. no fast message can overtake a slow progressing one. This definition closely emulates the standard definition used for asynchronous distributed algorithms [12].

Obviously, any algorithm that works correctly in the asynchronous mode will also work correctly in the synchronous mode, but the converse is *not* generally true: extra care may be needed to transform a correct synchronous algorithm into a correct asynchronous one; there are also general control layers, such as *synchronisers*, that can attempt to run a synchronous algorithm on an existing asynchronous runtime, but this does not always work [12].

4 Multi-cell P System (synchronous)

This model is based on *inter-cell parallelism* and assumes that the image pixels are distributed over a rectangular grid of cells, one cell per pixel, linked into a graph corresponding to the “standard” 8-neighbourhood. Figure 2a describes this layout, with arc labels, from center pixel to neighbours and from neighbours to center pixel.

Numeric complex cells contents are given as multisets over the unary base \circ , i.e. $\lambda = 0$, $\circ = 1$, $\circ\circ = \circ^2 = 2$, ...

Each cell starts in state S_0 , with the following initial values:

- One (immutable) ID symbol, $\iota(I, J)$, which indicates its coordinates: $I =$ row number, $J =$ column number.
- One (mutable) symbol, $p(C)$, which indicates its current colour: $C = b =$ black, or $C = w =$ white. Intuitively, p_0 has colour C .
- A multiset of eight (immutable) symbols, $n(X, Y)$, representing a “table”, which codifies the neighbourhood relation, from the current cell’s point of view, e.g. my neighbour X (e.g. 1=NW) knows me as neighbour Y (e.g. 5=SE):

$$\begin{array}{ccc} n(1, 5) & n(2, 6) & n(3, 7) \\ n(8, 4) & & n(4, 8) \\ n(7, 3) & n(6, 2) & n(5, 1) \end{array}$$

- A multiset of 2^8 (immutable) symbols, $\psi(\dots)$, representing a “table”, which completely defines the namesake table mentioned in Section 2, e.g.

$$\begin{array}{l} \psi(0, 0, 0, 0, 0, 0, 0, 0) \dots \\ \psi(0, 1, 1, 1, 1, 1, 0, 0, 1) \\ \psi(0, 1, 1, 1, 1, 1, 0, 1, 0) \dots \\ \psi(1, 1, 1, 1, 1, 1, 1, 1, 0) \end{array}$$

- One (immutable) symbol, $a(H)$, where $H \in \{\lambda, \circ\} = \{0, 1\}$, representing a target checkerboard marker, which is used to determine the current activity status:
 - If $H = \lambda = 0$, then initially cells $(0, 0)$, $(1, 1)$ are active, and cells $(1, 0)$, $(0, 1)$ are inactive.
 - If $H = \circ = 1$, then initially cells $(0, 0)$, $(1, 1)$ are inactive, and cells $(1, 0)$, $(0, 1)$ are active.
- A multiset of two (immutable) symbols, $\eta(X, Y)$, representing a “table”, which codifies the checkerboard flip:

$$\eta(\lambda, \circ) \quad \eta(\circ, \lambda)$$

Other used (mutable and temporary) symbols:

- Symbol $h(H)$ is the current cell’s activity marker and flips at each iteration phase, using table η . The activity status is determined by testing $h(\dots)$ against the target activity marker, $a(\dots)$: if both markers have the same contents, i.e. $h(H) \wedge a(H)$, then the cell is currently active; otherwise, the cell is currently inactive.
- Symbol $p(X, C)$ indicates that my neighbour’s X colour is C , or, intuitively $p_X = C$.

The ruleset given in Figure 4, which is also schematically represented in the flowchart of Figure 5, contains 8 states and 14 generic rules. This ruleset models the pseudocode of Listing 1, enhanced with details required by the multi-cell message-based distributed memory system.

Essentially, the synchronisation barriers are implicit and helped (as needed) by idempotent rules, which only advance the state, without any content change. The colours are exchanged by “properly timed” *push notification* messages, which ensure that no “contamination” is possible between “old” and “new” colours.

Initially, because they do not know any of their neighbours’ colours, all cells (black and white) send their colour to all their neighbours. Later, only black cells that turn white send their new white colour to all their black neighbours (who might need this). This limits the number of exchanged messages to the minimum possible. The given ruleset does not include (i) the termination check, which could be done e.g. by using an extra master cell; nor does it include (ii) any post-termination cleaning (here all cells end with a full set of their final neighbours’ colours).

The Ψ rules 11 and 12 use a promoter designated (for brevity) by a meta-syntactic abbreviation, Ψ , which is defined as follows:

$$\Psi = p(1, C_1) p(2, C_2) \dots p(8, C_8) \\ \psi(C_1, C_2, \dots, C_8, 1)$$

This rule checks the “deletion” condition of Section 2, $\psi = 1$, using its namesake table and the colours received from the eight neighbours.

$$\begin{aligned}
S_0 &\rightarrow_{\min\otimes\min} S_1 \mid p(w) & (1) \\
S_0 &\rightarrow_{\min\otimes\min} S_{11} \ h(IJ) \mid \iota(I, J) & (2) \\
S_1 &\rightarrow_{\max\otimes\min} S_2 \ (p(Y, w)) \downarrow_X \mid n(X, Y) & (3) \\
S_2 &\rightarrow_{\min\otimes\min} S_2 & (4) \\
S_{11} \ h/(\circ\circ) &\rightarrow_{\min\otimes\max} S_{12} \ h/\lambda & (5) \\
S_{11} &\rightarrow_{\max\otimes\min} S_{12} \ (p(Y, b)) \downarrow_X \mid n(X, Y) & (6) \\
S_{12} \ p(X, w) \ p(X, b) &\rightarrow_{\max\otimes\min} S_{13} \ p(X, w) & (7) \\
S_{12} \ h(H) &\rightarrow_{\min\otimes\min} S_{13} \ h(\bar{H}) \mid \eta(H, \bar{H}) & (8) \\
S_{13} &\rightarrow_{\min\otimes\min} S_{14} \mid h(H) \ a(H) & (9) \\
S_{13} &\rightarrow_{\min\otimes\min} S_{34} & (10) \\
S_{14} \ p(b) &\rightarrow_{\min\otimes\min} S_2 \ p(w) \mid \Psi & (11) \\
S_{14} &\rightarrow_{\max\otimes\min} S_2 \ (p(Y, w)) \downarrow_X \mid n(X, Y) \ \Psi & (12) \\
S_{14} &\rightarrow_{\min\otimes\min} S_{12} & (13) \\
S_{34} &\rightarrow_{\min\otimes\min} S_{12} & (14)
\end{aligned}$$

Figure 4: Synchronous P ruleset for the multi-cell scenario

From the start state, S_0 , cells follow two separate branches: (rule 1) white cells go to state S_1 ; while (rule 2) black cells go to state S_{11} and also start to compute their activity marker, h .

At state S_1 , white cells: (rule 3) send white colour notifications to all their neighbours (all unknown at this stage); and then continue to state S_2 , where (rule 4) they cycle indefinitely. This branch is missing in the high-level pseudocode of Listing 1, but required in our message-based distributed memory model.

At state S_{11} , black cells: (rule 5) collapse the content of their activity markers, h , to at most one single \circ ; (rule 6) send their black colour notifications to all their neighbours (all unknown at this stage); and then continue to state S_{12} .

State S_{12} is the start of the main black cell cycle, corresponding to the **while** line of Listing 1. Here, this cycle takes exactly *three* P steps and corresponds to *one* logical algorithm iteration. At state S_{12} , black cells: (rule 7) update their received colour notifications, in case previous neighbouring black cells have turned white; (rule 8) flip their activity indicator, h ; and then continue to state S_{13} .

State S_{13} corresponds to the activity checking **if** line of Listing 1. Here, black cells follow two separate branches: (rule 9, **then** branch) active black cells continue to state S_{14} ; while (rule 10, **else** branch) inactive black cells continue to state S_{34} .

At state S_{14} , active black cells take a decision based on the combined “deletion” condition, Ψ . Active black cells that validate Ψ (**if then** branch of Listing 1): (rule 11) turn white; (rule 12) send new (white) colour notifications to all their neighbours; and then *break* the cycle and continue to state S_2 . Active black cells that remain black (rule 13) return to state S_{12} (to start a new iteration).

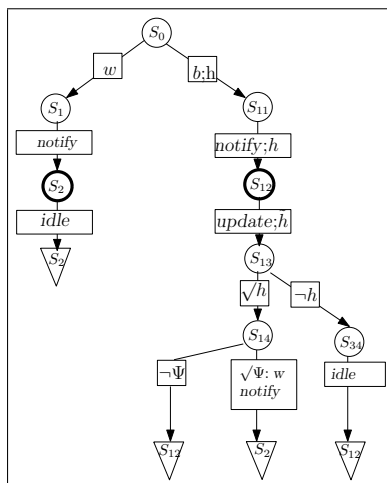


Figure 5: Schematic flowchart for the multi-cell ruleset

At state S_{34} , inactive black cells: (rule 14) take an idempotent “idle” step, required for proper synchronisation; and then go back to state S_{12} (to start a new iteration). Note that, at state S_{12} , black cells flip their activity indicator: thus, previously active black cells become inactive and vice-versa.

This analysis leads to the following proposition:

Theorem 4.1. The multi-cell system, constructed and initialised as discussed above, correctly models Guo and Hall’s parallel thinning algorithm.

Remark 4.2. For clarity, this ruleset does not include rules to check the algorithm termination and then to change state to an idle final state. This can be done in several ways; perhaps the simplest practical way is to set up a dedicated master cell, which can centralize the number of changes of each logical iteration.

Unless such extra check is done, under the traditional termination rules, the cells will not know the termination and continue to run their cycles, but without changing their essential content, which is the colour.

One could investigate a more relaxed theoretical termination condition, which could say that a system terminates when it enters an endless cycle which exactly repeats the same cell states and contents.

Could the cells themselves detect this, running a control layer based on a combination of a distributed termination detection algorithm with a cycle detection algorithm, such as Floyd’s “Tortoise and Hare” algorithm [7]? This seems to be an open question.

5 Single-cell P System

The design is similar to the multi-cell synchronous model of Section 4. A single-cell system is automatically synchronous, so again there is no need for explicit synchronisation barriers.

The single-cell model uses one single cell and maps each pixel (i, j) to a sub-cellular structure (i.e. a new complex object), with a carried functor, $\sigma(i, j)(\dots)$.

In the multi-cell model, each cell had its own state object, which controlled its lifeline. In the single-cell model, we simulate these states by local symbols in each sub-cell $\sigma(i, j)$: for consistency, we replace state S_i by symbol s_i and ensure that each sub-cell $\sigma(i, j)$ contains, at all times, exactly one such “state” symbol. Thus, this single-cell model is state-less.

The immutable target checkerboard symbol $a(H)$ and the immutable pre-computed tables, $n(X, Y)$, $\eta(H, \bar{H})$, $\psi(\dots)$, appear as top-level complex objects inside the single cell. Each $\sigma(i, j)(\dots)$ sub-cell contains a multiset which includes its own mutable colour symbol $p(C)$ and its own copies of the temporary mutable symbols used by the multi-cell model; the previous $\iota(i, j)$ ID is integrated as σ 's first parameter list.

The ruleset, listed in Figure 6, is similar to the ruleset of the multi-cell model of Figure 4, but adapted to work in parallel on all $\sigma(i, j)(\dots)$ sub-cells.

$$\begin{aligned}
\sigma(I, J) :: & & s_0 \rightarrow_{\max \otimes \min} s_1 \mid p(w) & (1) \\
\sigma(I, J) :: & & s_0 \rightarrow_{\max \otimes \min} s_{11} \mid h(IJ) & (2) \\
\sigma(I, J) :: & & s_1 \rightarrow_{\max \otimes \min} s_2 & (3) \\
\sigma(I, J) :: & & s_2 \rightarrow_{\max \otimes \min} s_2 & (4) \\
\sigma(I, J) :: & & h/(\circ\circ) \rightarrow_{\max \otimes \max} h/\lambda \mid s_{11} & (5) \\
\sigma(I, J) :: & & s_{11} \rightarrow_{\max \otimes \min} s_{12} & (6) \\
\sigma(I, J) :: & p(X, w) \ p(X, b) \rightarrow_{\max \otimes \min} p(X, w) \mid s_{12} & (7) \\
\sigma(I, J) :: & s_{12} \ h(H) \rightarrow_{\max \otimes \min} s_{13} \ h(\bar{H}) \mid / \eta(H, \bar{H}) & (8) \\
\sigma(I, J) :: & & s_{13} \rightarrow_{\max \otimes \min} s_{14} \mid h(H) / a(H) & (9) \\
\sigma(I, J) :: & & s_{13} \rightarrow_{\max \otimes \min} s_{34} & (10) \\
\sigma(I \circ, J \circ) :: & s_{14} \ p(b) \rightarrow_{\max \otimes \min} s_2 \ p(w) \mid \Psi' & (11) \\
\sigma(I \circ, J \circ) :: & & s_{14} \rightarrow_{\max \otimes \min} s_2 \mid \Psi' \quad \text{-not required-} & (12) \\
\sigma(I, J) :: & & s_{14} \rightarrow_{\max \otimes \min} s_{12} & (13) \\
\sigma(I, J) :: & & s_{34} \rightarrow_{\max \otimes \min} s_{12} & (14)
\end{aligned}$$

Figure 6: Synchronous P ruleset for the single-cell scenario.

Essential general changes:

- All rules (except the “ Ψ ” rules 11 and 12, which are further discussed below) are qualified by an access path identifying the current sub-cell, $\sigma(I, J) ::$, which also removes the need for $\iota(\dots)$ promoters
- Global items are qualified by the root access path: /
- All **min** instantiation modes are changed to **max**, which ensures that all $\sigma(I, J)$ sub-cells evolve in parallel
- States S_i are replaced by local symbols s_i , exactly one in each sub-cell $\sigma(I, J)$
- Rules for sending inter-cell colour notification messages are omitted (as we

have now direct access to corresponding contents of the required “neighbouring” $\sigma(\dots)$ sub-cells).

For example, (multi-cell) rule 2 of Figure 4:

$$S_0 \rightarrow_{\min \otimes \min} S_{11} \ h(IJ) \mid \iota(I, J) \ p(b)$$

is changed to:

$$\sigma(I, J)/s_0 \rightarrow_{\max \otimes \min} \sigma(I, J)/s_{11} \ \sigma(I, J)/h(IJ) \mid \sigma(I, J)/p(b)$$

or, equivalently, to rule 2 of Figure 6:

$$\sigma(I, J) :: s_0 \rightarrow_{\max \otimes \min} s_{11} \ h(IJ) \mid p(b)$$

Note that this rule: (i) will be instantiated once for each sub-cell $\sigma(I, J)$ and all its instances will run in parallel; (ii) uses only local $\sigma(I, J)(\dots)$ objects, such as s_i , $h(\dots)$ and $p(\dots)$, and there is no need for the $\iota(I, J)$ promoter, which is directly subsumed by the sub-cell itself, $\sigma(I, J)$.

Multi-cell rules 3, 5, 6, 7 and 12 of Figure 4 need special consideration. Besides now redundant colour notifications, rule 3, 6 and 12 ensured proper state transfers. Examining the context, we conclude that we only need corresponding “state” transfer rules for rules 3 and 6, but not for rule 12 (which is left in the listing, but could safely be omitted).

Rules 5 and 7 can perform multiple operations on the same sub-cell $\sigma(I, J)$. Therefore, the corresponding singleton “state” symbols, s_{11} and s_{12} , are transformed as promoters; subsequent rules, 6 and 8, are enough to ensure the required “state” transfers.

The “ Ψ ” transformations of the original rules 11 and 12 must offer fast access: (i) to “table” ψ , which is now global, and (ii) to the colours of “neighbouring” σ sub-cells, i.e. to all σ sub-cells which have coordinates in the range of plus/minus one from current coordinate. Therefore, these rules and all their objects (including all objects used in the Ψ' condition) use “custom” access paths, where I and J represent now the coordinates of the NW neighbour (i.e. current coordinates minus 1). Under this arrangement, rules 11 and 12 are not applicable to border sub-cells: e.g. the “top-left” sub-cell, $\sigma(\lambda, \lambda)$, cannot match a $\sigma(I\circ, J\circ)$ term – intuitively, because it does not have any “neighbours” on its left or above. However, this is acceptable, under our initial simplifying assumption that all borders pixels are white from start.

Let us recall that (multi-cell) rules 11 and 12 of Figure 4 tested the following condition, Ψ :

$$\begin{aligned} \Psi = & \ p(1, C_1) \ p(2, C_2) \ \dots \ p(8, C_8) \\ & \ \psi(C_1, C_2, \dots, C_8, 1) \end{aligned}$$

The colour notification rule 12 is not anymore needed, and (single-cell) rule 11 of Figure 6 tests now the following path qualified modified condition, Ψ' :

$$\begin{aligned} \Psi' = & \ / \sigma(I, J)/p(C_1) \quad / \sigma(I\circ, J)/p(C_2) \quad / \sigma(I\circ\circ, J)/p(C_3) \\ & \ / \sigma(I, J\circ)/p(C_8) \quad \quad \quad / \sigma(I\circ\circ, J\circ)/p(C_4) \\ & \ / \sigma(I, J\circ\circ)/p(C_7) \ / \sigma(I\circ, J\circ\circ)/p(C_6) \ / \sigma(I\circ\circ, J\circ\circ)/p(C_5) \\ & \ / \psi(C_1, C_2, \dots, C_8, 1) \end{aligned}$$

Theorem 5.1. The single-cell system, constructed and initialised as discussed above, correctly models Guo and Hall’s parallel thinning algorithm.

6 Multi-cell P System (asynchronous)

We first transform our *synchronous* multi-cell P system into a system that can also run in the *asynchronous* mode. Synchronous systems are easier to design, however, synchronous messages may slow down the evolution, may create deadlocks and may be difficult to map on Actors-based systems. For this algorithm, we have a straightforward ad-hoc way to transform it into an asynchronous version. Essentially, we ensure that each black cell knows exactly how many colour notifications are expected from other black cells, so it can cycle until all its due colour notifications are received.

Specifically, a black cell knows exactly how many colour notifications to expect, provided that all black cells continue to notify their neighbours at each iteration, even if they did not change colour. If I have received k black colour notifications from neighbours that were black at the previous logical step, then, at the next iteration, I wait until I see $k = k_b + k_w$ new colour notifications, where (i) k_b come from black cells that remained black (either they were inactive or they failed the Ψ test), and (ii) k_w come from active black cells that turned white. There is a cost for this, of course, as the cells now exchange many more colour notifications; however, forcing a system to run synchronously would also involve a non-negligible synchronisation cost, esp. in a true distributed setting.

The ruleset of the asynchronous version, listed in Figure 7, is similar to the ruleset of the synchronous model of Figure 4. To enable a straightforward comparison, we kept the original rule numbers; rules that have been expanded have additional numbers with letter suffixes: e.g. the synchronous rule 2 was expanded into asynchronous rules 2a and 2b.

We redefine the meaning of one symbol and we use two more, all indicating a colour C notified from neighbour X :

- $p(X, C)$: is now a received, but not yet recorded colour notification
- $q(X, C)$: is a recorded $p(X, C)$ colour notification
- $r(X, C)$: is a temporary stage of such a colour notification (between received and recorded)

Let us briefly discuss the new or enhanced rules of this asynchronous version, against corresponding “old” rules for the synchronous model. Rule 2a is identical to old rule 2 and is now complemented by rule 2b, which initialises each black cell with 8 “received” black notifications $q(X, b)$, $X \in [1, 8]$. As we don’t know anything yet about our neighbours, it is safe to initially assume that all our neighbours are black: all of them, black and white, will soon notify us their true colours.

Old rule 7 updates our recordings, for neighbours that were recorded black and had just notified us of their changed colour, and then goes to state S_{13} . In the synchronous setting, all such colour notifications arrived at the same time. Now, in an asynchronous setting, we cannot rush to state S_{13} , unless we have received all colour notifications, which may arrive with arbitrary delays. Therefore, rules 7a and 7b implement a small loop, trapping us in state S_{12} until we have received p colour notifications from all recorded black neighbours. For proper recording, we first keep the updated colours in temporary r objects and we only change these to recorded q objects, when we break out of this small cycle and go to state S_{13} , by rule 7c.

$$\begin{aligned}
S_0 &\rightarrow_{\min\otimes\min} S_1 \mid p(w) & (1) \\
S_0 &\rightarrow_{\min\otimes\min} S_{11} \mid h(IJ) \mid \iota(I, J) & (2a) \quad (2) \\
S_0 &\rightarrow_{\max\otimes\min} S_{11} \mid q(X, b) \mid n(X, -) & (2b) \\
S_1 &\rightarrow_{\max\otimes\min} S_2 \mid (p(Y, w)) \uparrow_X \mid n(X, Y) & (3) \\
S_2 &\rightarrow_{\min\otimes\min} S_2 & (4) \\
S_{11} \mid h/(o\circ) &\rightarrow_{\min\otimes\max} S_{12} \mid h/\lambda & (5) \\
S_{11} &\rightarrow_{\max\otimes\min} S_{12} \mid (p(Y, b)) \uparrow_X \mid n(X, Y) & (6) \\
S_{12} \mid p(X, C) \mid q(X, b) &\rightarrow_{\max\otimes\min} S_{12} \mid r(X, C) & (7a) \quad (7) \\
S_{12} &\rightarrow_{\min\otimes\min} S_{12} \mid q(-, b) & (7b) \\
S_{12} \mid r(X, C) &\rightarrow_{\max\otimes\min} S_{13} \mid q(X, C) & (7c) \\
S_{12} \mid h(H) &\rightarrow_{\min\otimes\min} S_{13} \mid h(\bar{H}) \mid \eta(H, \bar{H}) & (8) \\
S_{13} &\rightarrow_{\min\otimes\min} S_{14} \mid h(H) \mid a(H) & (9) \\
S_{13} &\rightarrow_{\min\otimes\min} S_{34} & (10) \\
S_{14} \mid p(b) &\rightarrow_{\min\otimes\min} S_2 \mid p(w) \mid \Psi'' & (11a) \quad (11) \\
S_{14} &\rightarrow_{\max\otimes\min} S_2 \mid (p(Y, w)) \uparrow_X \mid n(X, Y) \mid \Psi'' & (12a) \quad (12) \\
S_{14} &\rightarrow_{\max\otimes\min} S_{12} \mid (p(Y, b)) \uparrow_X \mid n(X, Y) \mid q(X, b) & (13a) \quad (13) \\
S_{14} &\rightarrow_{\min\otimes\min} S_{12} & (13b) \\
S_{34} &\rightarrow_{\max\otimes\min} S_{12} \mid (p(Y, b)) \uparrow_X \mid n(X, Y) \mid q(X, b) & (14a) \quad (14) \\
S_{34} &\rightarrow_{\min\otimes\min} S_{12} & (14b)
\end{aligned}$$

Figure 7: Asynchronous P ruleset for the multi-cell scenario

Rules 11a and 12a use a slightly new version of the meta-syntactic abbreviation, Ψ'' , which is now defined in terms of recorded notifications, $q(\dots)$ (instead of $p(\dots)$):

$$\begin{aligned}
\Psi &= q(1, C_1) \mid q(2, C_2) \mid \dots \mid q(8, C_8) \\
&\quad \psi(C_1, C_2, \dots, C_8, 1)
\end{aligned}$$

Rules 13a and 14a, the new versions of rules 13 and 14, ensure that now even black cells that remain black still send black colour notifications to their recorded black neighbours – this is required for our local synchronisation. Rules 13b and 14b ensure that black cells still go to the loop state S_{12} , even if they have no black neighbours; obviously these rules are not really required: these rules are here just to maintain full compatibility with the other versions and the idea that the system loops until it is properly terminated.

Example 6.1. Let us consider the first evolutionary steps of a cell, $\sigma(i\circ, j\circ)$, corresponding to the centre pixel of configuration #125 of Figure 3b. Rule 2b initialises $\sigma(i\circ, j\circ)$'s contents with 8 “received” q colour notifications: $q(1, b)$, $q(2, b)$, $q(3, b)$, $q(4, b)$, $q(5, b)$, $q(6, b)$, $q(7, b)$, $q(8, b)$, which indicate that, at the next logical step, $\sigma(i\circ, j\circ)$ expects 8 colour notifications (updates).

Next, all cells send their initial colours: white cells by rule 3 and black cells by rule 6. Cell $\sigma(i\circ, j\circ)$ cycles on state S_{12} until it receives 8 distinct $p(X, C)$

notifications, for $X \in [1, 8]$. For example, assume that σ receives, in *order*, the following notifications (including one ahead of time from its NE neighbour # 3, $\sigma(i \circ \circ, j)$): $p(3, b)$, $p(4, b)$, $p(1, b)$, $p(8, w)$, $p(2, w)$, $p(6, b)$, $p(7, b)$, $p(3, w)$, $p(5, b)$.

Note that the receiving order is important, as $\sigma(i \circ, j \circ)$ first considers $p(3, b)$ and keeps $p(3, w)$ for the next logical iteration. Then, as all expected notifications have been received, and $\sigma(i \circ, j \circ)$'s contents will finally record the following colours: $q(1, b)$, $q(2, w)$, $q(3, b)$, $q(4, b)$, $q(5, b)$, $q(6, b)$, $q(7, b)$, $q(8, w)$, $p(3, w)$.

Next, $\sigma(i \circ, j \circ)$ will proceed to check its activity marker, and, if active, will test the Ψ'' condition on its received notifications, q , and decide to remain black; in any case its will send again black colour notifications to all its neighbours, either by rule 13 or by rule 14; etc. Note the ahead-of-time received notification from its NE neighbour, $p(3, w)$, is now already waiting to be processed.

7 Actors Emulation

Traditional Actor systems can run efficiently on multi-cores, and there are novel extensions that extend Actors to heterogeneous platforms, involving clusters of nodes, with both CPU multi-core and GPU many-cores facilities. Therefore, mapping P systems to Actor based systems may prove to be a very promising emulation path.

Actor messaging is typically asynchronous, although some systems do also support some sort of synchronous messaging; for more details, please see the mentioned Actors references, and, specifically for F#, Syme's monograph [24].

A straightforward (but not the most efficient) Actors mapping of the asynchronous multi-cell model uses one actor for each cell (pixel) σ . All contents of σ become local variables in the corresponding actor. All P systems messaging, i.e. all colour notifications, is implemented as asynchronous messaging between actors. In F#, the lifetime of an actor is described by a sequence of mutually tail recursive async monad instances. A straightforward (but, again, not the most efficient) implementation of the ruleset maps each state to an async returning function. Leaving aside (as it would require too much additional space) part of the setup and a few other practical details, Figures 8 and 9 suggest the essential structure of such a cell actor.

A straightforward Actors mapping of the single-cell model uses one actor for each sub-cell (pixel) $\sigma(i, j)$. In this design, local contents of $\sigma(i, j)$ become local variables in the corresponding actor. All other top-level objects become global objects, accessible to all actors in the emulation. Because of these global shared objects, this is not a pure actor design, but, if properly designed, it can be both correct and efficient. Also, in this scenario, there need not be any messaging between actors, so actors can be replaced by more mundane tasks. However, if there are no messages that could be used for synchronisation, the system needs synchronisation barriers, as indicated in Section 2.

An interesting design (not developed yet) can combine these two designs, multi-cell and single-cell, for P system models and for their actor based implementations. One could partition the image in smaller rectangles and allocate sub-images to actors, perhaps distributed on different nodes or using different computing devices, such as CPUs vs GPUs. Such a partitioned design could be efficient for processing very large images, when running on heterogeneous clusters; however, it could also improve the efficiency on single multi-core nodes, by

```

type Colour = | White = 0 | Black = 1 | None = 2

type ReceivedColour = int * Colour

let N = [| 0; 5; 6; 7; 8; 1; 2; 3; 4 |]
let DI = [| 0; -1; 0; 1; 1; 1; 0; -1; -1 |]
let DJ = [| 0; -1; -1; -1; 0; 1; 1; 1; 0 |]

let neighbour (i:int, j:int, x:int) =
    (i + DI.[x], j + DJ.[x])

type PixelActor = option<MailboxProcessor<ReceivedColour>>

let N1 = 100
let N2 = 100

```

Figure 8: Simplified cell/pixel actor setup.

reducing the numbers of actors and thus eliminating some housekeeping overhead.

8 Conclusions

We present three membrane computing models for a complex image processing task – image skeletonisation: (i) a synchronous model using one cell per pixel and relying on inter-cell parallelism; (ii) an asynchronous model using one cell per pixel and relying on inter-cell parallelism; (iii) a (synchronous) model using one single cell, with one sub-cellular object per pixel, relying on intra-cell parallelism.

All three models are crisp, use reasonably small fixed-sized alphabets and rulesets, and are closely inter-related by way of almost mechanical translations. This experience has enabled us to further validate our complex objects proposal and enhance it with a “micro-surgery” facility, very useful for nested objects. The proposed models can be straightforwardly translated and implemented in a functional language with an Actors library.

Further work needs to be done to investigate the possible automation of these translations: (i) between multi-cell and single-cell P systems; (ii) from synchronous to asynchronous P systems; (iii) from P systems to Actors. Also, further work seems necessary to study heterogeneous designs, mixing ideas from multi-cell, single-cell, synchronous and asynchronous designs.

This study also suggests a couple of open questions: (i) to investigate partial asynchronous versions of similar algorithms and their merits (with respect to quality of results and runtime performance); (ii) to investigate an efficient distributed termination detection control layer, which is adequate for this algorithm (and other similar non-diffusing algorithms); possibly combining a distributed termination algorithm with a cycle detection algorithm.

Acknowledgments. Thanks to Zhengping Wang and to the anonymous reviewers for their most valuable comments and suggestions.

References

- [1] G. Agha. An algebraic theory of Actors and its application to a simple object-based language. In *In Ole-Johan Dahls Festschrift, volume 2635 of LNCS*, pages 26–57. Springer, 2004.
- [2] T. Bălănescu, R. Nicolescu, and H. Wu. Asynchronous P systems. *International Journal of Natural Computing Research*, 2(2):1–18, 2011.
- [3] D. Díaz-Pernil, F. Peña-Cantillana, and M. A. Gutiérrez-Naranjo. A parallel algorithm for skeletonizing images by using spiking neural P systems. *Neurocomputing*, 115:81–91, 2013.
- [4] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. A faster P solution for the Byzantine agreement problem. In M. Gheorghe, T. Hinze, and G. Păun, editors, *Conference on Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 175–197. Springer-Verlag, Berlin Heidelberg, 2010.
- [5] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. P systems and the Byzantine agreement. *Journal of Logic and Algebraic Programming*, 79(6):334–349, 2010.
- [6] H. ElGindy, R. Nicolescu, and H. Wu. Fast distributed DFS solutions for edge-disjoint paths in digraphs. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, and G. Vaszil, editors, *Membrane Computing*, volume 7762 of *Lecture Notes in Computer Science*, pages 173–194. Springer Berlin Heidelberg, 2013.
- [7] R. W. Floyd. Nondeterministic algorithms. *J. ACM*, 14(4):636–644, Oct. 1967.
- [8] G. Gimelfarb, R. Nicolescu, and S. Ragavan. P systems in stereo matching. In P. Real, D. Díaz-Pernil, H. Molina-Abril, A. Berciano, and W. Kropatsch, editors, *Computer Analysis of Images and Patterns*, volume 6855 of *Lecture Notes in Computer Science*, pages 285–292. Springer Berlin Heidelberg, 2011.
- [9] G. Gimelfarb, R. Nicolescu, and S. Ragavan. P system implementation of dynamic programming stereo. *Journal of Mathematical Imaging and Vision*, 47(1-2):13–26, 2013.
- [10] Z. Guo and R. W. Hall. Parallel thinning with two-subiteration algorithms. *Commun. ACM*, 32(3):359–373, Mar. 1989.
- [11] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323 – 364, 1977.
- [12] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
- [13] R. Nicolescu. Parallel and distributed algorithms in P systems. In M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, editors, *Membrane Computing, CMC 2011, Revised Selected Papers*, volume 7184 of

- Lecture Notes in Computer Science*, pages 35–50. Springer Berlin / Heidelberg, 2012.
- [14] R. Nicolescu, G. Gimelfarb, J. Morris, R. Gong, and P. Delmas. Regularising ill-posed discrete optimisation: Quests with P systems. *Fundam. Inf.*, 131(3-4):465–483, 2014.
 - [15] R. Nicolescu, F. Ipate, and H. Wu. Programming P systems with complex objects. In A. Alhazov, S. Cojocaru, M. Gheorghe, Y. Rogozhin, G. Rozenberg, and A. Salomaa, editors, *14th Conference on Membrane Computing, Revised Selected Papers*, volume 8340 of *Lecture Notes in Computer Science*, pages 280–300. Springer, 2013.
 - [16] R. Nicolescu, F. Ipate, and H. Wu. Towards high-level P systems programming using complex objects. In A. Alhazov, S. Cojocaru, M. Gheorghe, and Y. Rogozhin, editors, *14th International Conference on Membrane Computing, CMC14, Chişinău, Moldova, August 20-23, 2013, Proceedings*, pages 255–276. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Chişinău, 2013.
 - [17] R. Nicolescu and H. Wu. BFS solution for disjoint paths in P systems. In C. Calude, J. Kari, I. Petre, and G. Rozenberg, editors, *Unconventional Computation*, volume 6714 of *Lecture Notes in Computer Science*, pages 164–176. Springer Berlin Heidelberg, 2011.
 - [18] R. Nicolescu and H. Wu. New solutions for disjoint paths in P systems. *Natural Computing*, 11:637–651, 2012.
 - [19] R. Nicolescu and H. Wu. Complex objects for complex applications. *Romanian Journal of Information Science and Technology*, (to appear), 2014.
 - [20] F. Peña-Cantillana, A. Berciano, D. Díaz-Pernil, and M. A. Gutiérrez-Naranjo. Parallel skeletonizing of digital images by using cellular automata. In M. Ferri, P. Frosini, C. Landi, A. Cerri, and B. D. Fabio, editors, *CTIC*, volume 7309 of *Lecture Notes in Computer Science*, pages 39–48. Springer, 2012.
 - [21] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
 - [22] R. Reina-Molina and D. Díaz-Pernil. Bioinspired parallel 2D or 3D skeletonization. *IMAGEN-A*, 3(5):41–44, 2013.
 - [23] R. Reina-Molina, D. Díaz-Pernil, and M. A. Gutiérrez-Naranjo. Cell complexes and membrane computing for thinning 2D and 3D images. In M. A. M. del Amor, G. Păun, I. Pérez-Hurtado, and F. J. Romero-Campero, editors, *Tenth Brainstorming Week on Membrane Computing*, volume 1 of *RGNC REPORT*, pages 91–110. Universidad de Sevilla, 2012.
 - [24] D. Syme, A. Granicz, and A. Cisternino. *Expert F# 3.0*. Apress, Berkely, CA, USA, 3rd edition, 2012.

- [25] H. Wu. Minimum spanning tree in P systems. In L. Pan, G. Păun, and T. Song, editors, *Proceedings of the Asian Conference on Membrane Computing (ACMC2012)*, pages 88–104, Wuhan, China, 2012. Huazhong University of Science and Technology.
- [26] H. Wu. *Distributed Algorithms in P Systems*. PhD thesis, The University of Auckland, Auckland, New Zealand, 2014.

```

let SetPixel (i:int, j:int, p:Colour) =
  MailboxProcessor<ReceivedColour>.Start (fun inbox ->
    let h = ref 0
    let P = Array.create 8 Colour.None
    let Q = Array.create 8 Colour.Black

    let rec S0 () = async {
      match p with
      | Colour.White ->
        return! S1 ()
      | - ->
        h := i + j
        return! S11 ()
    }

    and S11 () = async {
      while !h >= 2 do
        h := !h - 2
      for x = 1 to 8 do
        let n = neighbour(i, j, x)
        Pixels.[fst n, snd n].Value.Post (N.[x], p)
      return! S12 ()
    }

    and S12 () = async {
      let R = Array.create 8 Colour.None

      for x = 1 to 8 do
        if P.[x] <> Colour.None then
          Q.[x] <- Colour.None
          R.[x] <- P.[x]

      while Array.exists (fun c -> c = Colour.Black) Q do
        let! m = inbox.Receive()
        match m with
        | (x, c) when Q.[x] = Colour.Black ->
          Q.[x] <- Colour.None
          R.[x] <- c
        | (x, c) -> // Q.[x] = Colour.None
          P.[x] <- c

      for x = 1 to 8 do
        if R.[x] <> Colour.None then
          Q.[x] <- R.[x]

      return! S13 ()
    }

    // ...

    S0 () )

```

Figure 9: Simplified cell/pixel actor body.

Causal nets for geometrical Gandy–Păun–Rozenberg machines

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences
Śniadeckich 8, 00-656 Warsaw, Poland
A.Obtulowicz@impan.pl

Abstract. An approach to the computational complexity beyond the known complexity measures of the consumed time and space of computation is proposed. The approach focuses on the chaotic behavior and randomness aspects of computational processes and is based on a representation of these processes by causal nets.

1 Introduction

A certain new approach to the investigations of the computational complexity of abstract systems allowing some unrestricted parallelism of computation is proposed, where the computational processes realized in a discrete time with a central clock by these systems are represented by causal nets similar to those in [4] and related to causal sets in [1].

The representation of computational processes by causal nets is aimed to provide an abstraction from those features of computational processes which do not have a spatial nature such that the abstraction could make visible some new aspects of the processes like an aspect of chaotic behavior or a fractal shape.

The aspects of a chaotic behavior and a fractal shape inspired by the research area of dynamics of nonlinear systems [20] regarding an unpredictability of the behavior of these systems¹ could suggest an answer to the following question formulated in [21]: *Is the concept of randomness, founded in the concept of absence of computable regularities, the only adequate and consistent one? In which direction, if any, should one look for alternatives?*

The answers may have an impact on designing pseudorandom number generators, cf. [23], [24], applied in statistics, cryptography, and Monte Carlo Method.

The proposed approach is aimed to provide a possibly uniform precise mathematical treatment of causal nets and related concepts which could serve for measuring of complexity of computational processes by a use of graph dimensions [13] and network fractal dimensions [19], [7], [8] in parallel to measuring complexity of random strings in [11] by Hausdorff dimension.

The proposed approach concerns the investigations of abstract computing devices which are geometrical Gandy–Păun–Rozenberg machines.

¹ unpredictability due to sensitive dependence on initial conditions—an important feature of deterministic transient chaos [20] often having fractal shape.

The geometrical Gandy–Păun–Rozenberg machines are some modifications of the known Gandy–Păun–Rozenberg machines [14], [15].

The assumption that the sets of instantaneous descriptions of geometrical Gandy–Păun–Rozenberg machines are skeletal sets of finite directed graphs together with the features of machine local causation rewriting rules provide a natural construction of causal nets representing computational processes.

2 Geometrical Gandy–Păun–Rozenberg machines

We refer the reader to Appendix A and Appendix B (quoting the main definitions of [14], [15]) for unexplained notions and notation concerning labelled directed graphs, Gandy–Păun–Rozenberg machines (briefly G–P–R machines), and generalized G–P–R machines.

We recall the main difference between G–P–R machines and generalized G–P–R machines:

- the auxiliary rules of the G–P–R machines are not specified and for every G–P–R machine \mathcal{M} with its transition function $\mathcal{F}_{\mathcal{M}}$ and for every instantaneous description G of \mathcal{M} the instantaneous description $\mathcal{F}_{\mathcal{M}}(G)$ is a colimit of the gluing diagram \mathcal{D}^G determined by the set $\mathcal{P}\ell(G)$ of maximal applications of the machine rewriting rules to G .
- the generalized G–P–R machines are equipped with auxiliary rules besides the rewriting rules and for every generalized G–P–R machine \mathcal{M} with its transition function $\mathcal{F}_{\mathcal{M}}$ and for every instantaneous description G of \mathcal{M} the instantaneous description $\mathcal{F}_{\mathcal{M}}(G)$ is a colimit of the generalized gluing diagram \mathcal{D}_G determined by both the machine rewriting rules and the auxiliary rules.

Definition 2.1. We define a *simple geometrical G–P–R machine* and a *strict geometrical G–P–R machine* to be the modifications of a G–P–R machine and a generalized G–P–R machine, respectively, such that

- in both cases of a simple and a strict machine we assume that
 - the set of labels of vertices of the directed graphs belonging to the set of instantaneous descriptions of a given machine is a one element set or equivalently these graphs are not labelled at all, an analogous assumption concerns the graphs appearing in the machine rewriting rules and auxiliary rules,
 - for every machine \mathcal{M} there exists a natural number $n > 0$ such that for every graph G belonging to the set of instantaneous descriptions of \mathcal{M} the set $V(G)$ of vertices of G is a set of ordered n -tuples of elements of Q^\bullet , where Q^\bullet is the set of rational numbers, if necessary, extended to the recursive real numbers which are linear combinations of $\sqrt{2}$, $\sqrt{3}$, etc. with rational coefficients (hence we use the adjective ‘geometrical’),

- in the case of a simple machine we impose a strengthening that the graphs belonging to the set of instantaneous descriptions of the machine or appearing in the conclusions of the machine rewriting rules are not necessarily isomorphically perfect graphs.

Theorem. *For both cases of a simple and of a strict geometrical G–P–R machine if the set of its instantaneous description is a recursive set, then the transition function of the machine is a computable function.*

Proof. We prove the theorem for the case of a simple geometrical G–P–R machine \mathcal{M} with its transition function $\mathcal{F}_{\mathcal{M}}$.

The following assignments are computable:

- the assignment to a finite gluing diagram its colimit constructed as in Appendix A in the domain of finite directed graphs,
- the assignment to an instantaneous description G of \mathcal{M} the set $\mathcal{P}\ell(G)$ of maximal applications of the rewriting rules of \mathcal{M} ,
- the assignment to an instantaneous description G of \mathcal{M} the gluing diagram \mathcal{D}^G which is determined by $\mathcal{P}\ell(G)$ in an effective way.

Hence the assignment to an instantaneous description G of \mathcal{M} the result of the construction of a colimit of the gluing diagram \mathcal{D}^G is also computable assignment. Therefore an effective search of that unique instantaneous description $G' = \mathcal{F}_{\mathcal{M}}(G)$ which is isomorphic to the above result of the construction of a colimit of the gluing diagram \mathcal{D}^G suffices for reaching $\mathcal{F}_{\mathcal{M}}(G)$ in an effective way. This effective search is provided by the assumption that the set of instantaneous descriptions of the machine \mathcal{M} is a recursive set. Thus the transition function $\mathcal{F}_{\mathcal{M}}$ is a computable function.

The proof of the theorem for the case of strict geometrical G–P–R machines is similar to the above proof.

Examples 2.1 (The simulation of cellular automata). The generalized G–P–R machine \mathcal{M}^{SGL} in [15] is an example of a strict geometrical G–P–R machine, where \mathcal{M}^{SGL} simulates the spatial and temporal behavior of a cellular automaton identified with the eastern expansion fragment of Conway’s *Game of life*.

We show now an example of a simple geometrical G–P–R machine which is aimed to simulate the behavior of one-dimensional cellular automaton with two cell states 0, 1 and with the rule 30 given by the formula

$$a_{i-1} \text{ xor } (a_i \text{ or } a_{i+1}),$$

cf. [23], [24], where xor is ‘exclusive or’. This simple geometrical G–P–R machine, denoted by \mathcal{M}^{30} , is defined in the following way.

The instantaneous descriptions and the rewriting rules of \mathcal{M}^{30} are defined by using the finite directed graphs cl_x^n (for an integer n and $x \in \{0, 1, !, \emptyset\}$) corresponding to the single cells for $x \in \{0, 1, !\}$, where cl_x^n are such that:

- the graph cl_{\emptyset}^n is the square

$$\begin{array}{ccc} (n, 1) & \longrightarrow & (n+1, 1) \\ \uparrow & & \uparrow \\ (n, 0) & \longrightarrow & (n+1, 0) \end{array}$$

together with the loop $((0, 0), (0, 0))$ in the case $n = 0$, and with the path from $(n+1, 0)$ to $(n, 1)$ containing three intermediate vertices $(n+1 - \frac{i}{4}, \frac{i}{4})$ with $\{1, 2, 3\}$,

- the graph cl_x^n for $x \in \{0, 1, !\}$ consists of:
 - the graph cl_{\emptyset}^n as a subgraph,
 - the edge $((n, 0), (n+1 - \frac{x+1}{4}, \frac{x+1}{4}))$ for $x \in \{0, 1\}$, indicating that the graph cl_x^n corresponds to a cell in state x and called an *edge indicating a state of a cell*,
 - the edge $((n, 0), (n+1 - \frac{3}{4}, \frac{3}{4}))$ for $x = !$.

An instantaneous description of \mathcal{M}^{30} is that graph G which is the graph union (cf. Appendix A)

$$\text{cl}_i^i \cup \left(\bigcup_{i < k < j} \text{cl}_{x_k}^k \right) \cup \text{cl}_i^j$$

for some integers $i < -1$, $j > 1$ and for some family $\text{cl}_{x_k}^k$ ($i < k < j$) with $x_k \in \{0, 1\}$ for all integers k such that $i < k < j$.

The rewriting rules of \mathcal{M}^{30} are given by

- $\text{cl}_i^1 \cup \text{cl}_j^2 \cup \text{cl}_k^3 \vdash \text{cl}_{\emptyset}^1 \cup \text{cl}_{\rho(i,j,k)}^2 \cup \text{cl}_{\emptyset}^3$ for $\{i, j, k\} \subseteq \{0, 1\}$, where $\rho(i, j, k) = i \text{ xor } (j \text{ or } k)$,
- $\text{cl}_i^2 \cup \text{cl}_j^3 \cup \text{cl}_k^4 \vdash \text{cl}_i^1 \cup \text{cl}_{\rho(0,0,j)}^2 \cup \text{cl}_{\rho(0,j,k)}^3 \cup \text{cl}_{\emptyset}^4$ for $\{j, k\} \subseteq \{0, 1\}$,
- $\text{cl}_i^1 \cup \text{cl}_j^2 \cup \text{cl}_i^3 \vdash \text{cl}_{\emptyset}^1 \cup \text{cl}_{\rho(i,j,0)}^2 \cup \text{cl}_{\rho(j,0,0)}^3 \cup \text{cl}_i^4$ for $\{i, j\} \subseteq \{0, 1\}$,
- the identity rule $\bullet 0 \vdash \bullet 0$, where $\bullet 0$ is the graph with single vertex 0 and with single edge which is a loop.

The graphs cl_j^2 and cl_j^3 appearing in the middle of the premises of the above rules are called the *centers* of these rules, respectively.

The one-dimensional cellular automata in [23], [24] and small Turing machines in [7], [8] can be simulated by simple G–P–R machines constructed in a similar way to the machine \mathcal{M}^{30} .

Example 2.2 (generation of the contours of the iterations of fractals). We show a simple geometrical G–P–R machine whose single rewriting rule serves for generating the contours of the iterations of Sierpiński gasket. This machine, denoted by $\mathcal{M}^{\text{Sierp}}$ is defined in the following way.

Let Δ be a directed graph given by

$$\begin{aligned} V(\Delta) &= \left\{ (0, 0), \left(\frac{1}{2}, \frac{\sqrt{3}}{2} \right), (1, 0) \right\}, \\ E(\Delta) &= \left\{ ((0, 0), (1, 0)), ((0, 0), \left(\frac{1}{2}, \frac{\sqrt{3}}{2} \right)), ((1, 0), \left(\frac{1}{2}, \frac{\sqrt{3}}{2} \right)) \right\}. \end{aligned}$$

The graph Δ is a contour of an equilateral triangle.

The instantaneous descriptions of $\mathcal{M}^{\text{Sierp}}$ are graphs Δ_n (for a natural number $n \geq 0$) defined inductively by

$$\begin{aligned}\Delta_0 &= (V(\Delta), E(\Delta) \cup \{((0, 1), (0, 1))\}), \\ \Delta_{n+1} &= \bigcup_{i \in \{1, 2, 3\}} f_i(\Delta_n),\end{aligned}$$

where f_1, f_2, f_3 are functions forming the iterated function system for Sierpiński gasket, cf. [18] and Appendix C, and for a directed graph G with $V(G) \subset Q^\bullet \times Q^\bullet$ $f_i(G)$ is a graph such that

$$\begin{aligned}V(f_i(G)) &= \{f_i(v) \mid v \in V(G)\}, \\ E(f_i(G)) &= \{(f_i(v), f_i(v')) \mid (v, v') \in E(G)\}.\end{aligned}$$

A unique rewriting rule of $\mathcal{M}^{\text{Sierp}}$ is of the form

$$\Delta_1 \vdash \Delta_2.$$

The graphs Δ_n ($n > 0$) are the contours of the iterations of Sierpiński gasket.

The similar simple G–P–R machines can be constructed for some other fractals determined by iteration function system, e.g. 3D Sierpiński gasket.

3 Causal nets of geometrical G–P–R machines

We propose some precise mathematical treatment of those concepts which express or explicate certain aspects and features of the computational processes realized by geometrical G–P–R machines and which can be investigated within ‘experimental mathematics’ by an analysis (sometimes heuristic) of the plots illustrating those concepts. The plots could be generated by computers like in [24].

Definitions 3.1. For both cases of a simple or of a strict geometrical G–P–R machine \mathcal{M} and an initial instantaneous description G of \mathcal{M} we define an *event with respect to G* to be an ordered pair (v, i) with $v \in V(\mathcal{F}_{\mathcal{M}}^i(G))$ for a natural number $i \geq 0$, where $\mathcal{F}_{\mathcal{M}}^0(G) = G$. Then we define a *full causal relation \prec_G with respect to G* to be a binary relation defined on the set $Ev(G)$ of events with respect to G given by

$$(v, i) \prec_G (v', i') \quad \text{iff} \quad i' = i + 1 \text{ and there exists } h \in \mathcal{P}\ell(\mathcal{F}_{\mathcal{M}}^i(G))$$

such that $v \in V(\text{im}(h))$ and $v' \in V(\text{im}(q_h))$ for the h -th canonical injection $q_h : \mathcal{R}_{\mathcal{M}}(\text{dom}(h)) \rightarrow \mathcal{F}_{\mathcal{M}}^{i+1}(G)$ into the colimit of the gluing diagram $\mathcal{D}_{\mathcal{F}_{\mathcal{M}}^i(G)}$ in the simple case and of the generalized gluing diagram $\mathcal{D}_{\mathcal{F}_{\mathcal{M}}^i(G)}$ in the strict case, where $\mathcal{R}_{\mathcal{M}}(\text{dom}(h))$ is the conclusion of the rewriting rule with the premise $\text{dom}(h)$. Thus the ordered pair $\mathcal{N}_G = (Ev(G), \prec_G)$ is called a *full causal net of \mathcal{M} with respect to G* .

The proper subnets of the full causal net \mathcal{N}_G with respect to G correspond to various aspects and features of the computation of \mathcal{M} starting with G .

For instance, in the case of Example 2.2 it is worth to consider a *causal net* $\mathcal{N}_G^{\text{gr}}$ of growth with *causal growth relation* \prec_G^{gr} given by

$$(v, i) \prec_G^{\text{gr}} (v', i') \quad \text{iff} \quad i' = i + 1 \text{ with } (v, i) \prec_G (v', i') \text{ and both } v \text{ and } v' \\ \text{are new in } \mathcal{F}_{\mathcal{M}}^i(G) \text{ and in } \mathcal{F}_{\mathcal{M}}^{i+1}(G), \text{ respectively, whenever } i > 0, \\ \text{otherwise } v' \text{ is new in } \mathcal{F}^{i+1}(G),$$

where a vertex x is new in $\mathcal{F}_{\mathcal{M}}^k(G)$ if $x \in V(\mathcal{F}_{\mathcal{M}}^k(G))$ and $x \notin V(\mathcal{F}_{\mathcal{M}}^{k-1}(G))$ for $k > 0$.

In the case of the machine in Example 2.2 the projection of $\mathcal{N}_{\Delta_0}^{\text{gr}}$ into the phase space $Q^\bullet \times Q^\bullet$ yields Sierpiński gasket which is a fractal.

In the case of Examples 2.1 it is worth to consider a *causal net* $\mathcal{N}_G^{\text{act}}$ of activity with *causal relation* \prec_G^{act} of activity given by

$$(v, i) \prec_G^{\text{act}} (v', i') \quad \text{iff} \quad i' = i + 1 \text{ with } (v, i) \prec_G (v', i') \text{ and both } v \text{ and } v' \\ \text{are the targets of the edges indicating the states} \\ \text{of the cells in } \mathcal{F}_{\mathcal{M}}^i(G) \text{ and in } \mathcal{F}_{\mathcal{M}}^{i+1}(G), \text{ respectively.}$$

For the geometrical G–P–R machines simulating one-dimensional cellular automata like the machine \mathcal{M}^{30} in Examples 2.1 one defines the *causal net* $\mathcal{N}_G^{\text{stc}}$ of strict changes with the *causal relation* \prec_G^{stc} of strict changes given by

$$(v, i) \prec_G^{\text{stc}} (v', i') \quad \text{iff} \quad i' = i + 1 \text{ and there exists } h \in \mathcal{P}\ell(\mathcal{F}_{\mathcal{M}}^i(G)) \\ \text{such that } h(v_1) = v \text{ and } q_h(v_2) = v' \text{ for those } v_1, v_2 \text{ which are such that} \\ v_1 \text{ is a vertex of the center of the rule } \text{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\text{dom}(h)) \\ \text{and both } v_1, v_2 \text{ are the targets of the edges indicating the state} \\ \text{of a cell in the premise and in the conclusion of the rule, respectively.}$$

Thus $\mathcal{N}_G^{\text{stc}}$ is a subnet of $\mathcal{N}_G^{\text{act}}$, moreover, in the case of \mathcal{M}^{30} the plots for the one-dimensional cellular automaton with the rule 30 in [23], [24] illustrate appropriate nets $\mathcal{N}_G^{\text{stc}}$.

The nets $\mathcal{N}_G^{\text{stc}}$ ($G \in \mathcal{S}_{\mathcal{M}}$) coincide with space-time diagrams in [7], [8], where these diagrams are subject of the investigations of computational complexity by using fractal dimension.

The transitive closures \prec_G^* , $(\prec_G^x)^*$ ($x \in \{\text{gr}, \text{act}, \text{stc}\}$) give rise to causal sets $\mathcal{C}_G = (Ev(G), \prec_G^*)$ and $\mathcal{C}_G^x = (Ev(G), (\prec_G^x)^*)$ ($x \in \{\text{gr}, \text{act}, \text{stc}\}$) whose logical aspects can be approached like in physics [12] or like in concurrency theory [2].

The investigations of machine $\mathcal{M}^{\text{Sierp}}$ defined in Example 2.2 suggest another approach to the idea of a causal net of a computation of a geometrical G–P–R machine which is introduced in the following definitions.

Definitions 3.2. For both cases of a simple or of a strict geometrical G–P–R machine \mathcal{M} and an initial instantaneous description G of \mathcal{M} we define a *rule application event with respect to G* to be an ordered pair (h, i) with $h \in \mathcal{P}\ell(\mathcal{F}_{\mathcal{M}}^i(G))$ for a natural number $i \geq 0$, where $\mathcal{F}_{\mathcal{M}}^0(G) = G$. Then we define a *rule application causal relation \succ_G^{app} with respect to G* to be a binary relation defined on the set $Ev^{\text{app}}(G)$ of the rule application events with respect to G given by

$$(h, i) \succ_G^{\text{app}} (h', i') \quad \text{iff} \quad i' = i + 1 \text{ and } \text{im}(h) \text{ is a subgraph of } \text{im}(q_{h'})$$

for the h' -th canonical injection $q_{h'} : \mathcal{R}_{\mathcal{M}}(\text{dom}(h')) \rightarrow \mathcal{F}_{\mathcal{M}}^{i+1}(G)$ into the colimit of the gluing diagram $\mathcal{D}^{\mathcal{F}_{\mathcal{M}}^i(G)}$ in the simple case and of the generalized gluing diagram $\mathcal{D}_{\mathcal{F}_{\mathcal{M}}^i(G)}$ in the strict case. Thus the ordered pair $\mathcal{N}_G^{\text{app}} = (Ev^{\text{app}}(G), \succ_G^{\text{app}})$ is called a *causal net of the rule application events with respect to G* .

For natural numbers $n > 0$ the *restrictions of $\mathcal{N}_G^{\text{app}}$ to n* , denoted by $\mathcal{N}_G^{\text{app}} \upharpoonright n$, are the ordered pairs $(Ev^{\text{app}}(G) \upharpoonright n, \succ_G^{\text{app}} \upharpoonright n)$ with $Ev^{\text{app}}(G) \upharpoonright n = \{(h, i) \in Ev^{\text{app}}(G) \mid i \leq n\}$, where $\succ_G^{\text{app}} \upharpoonright n$ is the restriction of \succ_G^{app} to $Ev^{\text{app}}(G) \upharpoonright n$.

Lemma 3.1. *Machine $\mathcal{M}^{\text{Sierp}}$ is such that for every rule application event (h, i) with respect to Δ_0 with $i \geq 0$ there exists a unique ordered triple (h_1, h_2, h_3) of elements of $\mathcal{P}\ell(\mathcal{F}_{\mathcal{M}^{\text{Sierp}}}^{i+1}(\Delta_0))$ such that the following condition holds:*

$$(\alpha) \quad (h_j, i + 1) \succ_{\Delta_0}^{\text{app}} (h, i) \text{ and } h_j = f_j \circ h \text{ for all } j \in \{1, 2, 3\},$$

where f_1, f_2, f_3 form the iteration function system for Sierpiński gasket, cf. Appendix C, and \circ denotes the composition of functions.

Proof. We prove the lemma by induction on i .

Corollary 3.1. *Causal net $\mathcal{N}_{\Delta_0}^{\text{app}}$ of the rule application events with respect to Δ_0 for machine $\mathcal{M}^{\text{Sierp}}$ is isomorphic to the (ternary) tree \mathbb{T} whose vertices are finite strings (including empty string) of digits in $\{1, 2, 3\}$, the edges are ordered pairs $(\Gamma j, \Gamma)$ for a finite string Γ and a digit $j \in \{1, 2, 3\}$, where the graph isomorphism $\text{iz} : \mathbb{T} \rightarrow \mathcal{N}_{\Delta_0}^{\text{app}}$ is defined inductively by*

- $\text{iz}(\text{empty string}) = (\text{id}_{\Delta_0}, 0)$, where id_{Δ_0} is the identity graph homomorphism on Δ_0 ,
- $\text{iz}(\Gamma j) = (h', \text{length}(\Gamma) + 1)$ for a unique h' which is the j -th element of a unique ordered triple which satisfies the condition (α) for that h for which $\text{iz}(h) = (h, \text{length}(\Gamma))$.

Proof. The corollary is a consequence of Lemma 3.1.

Corollary 3.2. *Machine $\mathcal{M}^{\text{Sierp}}$ is such that for every rule application event (h, i) with respect to Δ_0 for $i \geq 0$ the unique ordered triple (h_1, h_2, h_3) of elements of $\mathcal{P}\ell(\mathcal{F}_{\mathcal{M}^{\text{Sierp}}}^{i+1}(\Delta_0))$ satisfying the condition (α) for h determines a directed multi-hypergraph $\mathcal{G}_{(h, i)}$ (see Appendix A) whose set of hyperedges is the set*

$\{(h_1, i+1), (h_2, i+1), (h_3, i+1)\}$, the set of vertices is the union $\bigcup_{1 \leq j \leq 3} V(\text{im}(h_j))$ and the source and target functions s, t are given by

$$\begin{aligned} s((h_j, i+1)) &= \{h_j((0, 0)), h_j((1, 0))\}, \\ t((h_j, i+1)) &= \{h_j((\frac{1}{2}, \frac{\sqrt{3}}{2}))\} \quad \text{for all } j \in \{1, 2, 3\}. \end{aligned}$$

Moreover, for every rule application event (h, i) with respect to Δ_0 for $i \geq 0$ the directed multi-hypergraph $\mathcal{G}_{(h,i)}$ is isomorphic to the directed multi-hypergraph $\mathcal{G}_{(\text{id}_{\Delta_0}, 0)}$.

Proof. The corollary is a consequence of Lemma 3.1.

Remark 3.1. The directed multi-hypergraph $\mathcal{G}_{(h,i)}$ in Corollary 3.2 could model some interaction between the rule application events in the computation process of $\mathcal{M}^{\text{Sierp}}$ starting with Δ_0 and represented by $\mathcal{N}_{\Delta_0}^{\text{app}}$. This interaction could be a gluing pattern understood as in the main theorem of [16].

Remark 3.2. Since the multi-hyperedge membrane systems $\mathcal{S}_n^{\text{Sierp}}$ in [16] for $n \geq 0$ are aimed to display the self-similar structure of (the iterations of) Sierpiński gasket by using isomorphisms of directed multi-hypergraphs and net $\mathcal{N}_{\Delta_0}^{\text{app}}$ represents the computation process of machine $\mathcal{M}^{\text{Sierp}}$ starting with Δ_0 , one can see (in the light of Corollaries 3.1 and 3.2) that the self-similar structure (or form) of the contours of the iterations of Sierpiński gasket coincides² with (or simply is) the process of their generation by machine $\mathcal{M}^{\text{Sierp}}$. This coincidence is similar to the coincidence of *Nautilus* shell, illustrated in Fig. 1 in [22], with the process of its growth.

Final Remark 3.3. The author expects that the geometrical G–P–R machines and their extensions to higher dimensions could provide the mathematical foundations for *the atomic basis of biological symmetry and periodicity*³ due to Antonio Lima-da-Faria [9]. These foundations could explicate the links of cellular automata approach to complexity in biology in S. Wolfram’s *A New Kind of Science* with *Evolution without selection* [10] pointed out by B. Goertzel in his review of *A New Kind of Science* in [6].

Open problem One can define geometrical G–P–R machines whose instantaneous descriptions are finite graphs with vertices labelled by multisets and the machine rewriting rules contain multiset rewriting rules like in membrane computing [17].

How to extract in the case of these machines the counterparts of causal nets to be subject of measuring uncertainty via fractal dimension like e.g. in [7], [8].

² by Corollaries 3.1 and 3.2 the restrictions $\mathcal{N}_{\Delta_0}^{\text{app}} \upharpoonright n$ together with the directed hypergraphs $\mathcal{G}(h, i)$ provide a construction of multihyperedge membrane systems (with the restrictions $\mathcal{N}_{\Delta_0}^{\text{app}} \upharpoonright n$ as their underlying trees) isomorphic to $\mathcal{S}_n^{\text{Sierp}}$.

³ selfsimilarity characterized in terms of geometrical G–P–R machines like in $\mathcal{M}^{\text{Sierp}}$ case could be a counterpart of spatial periodicity with respect to both time and scale changes.

Appendix A. Graph-theoretical and category-theoretical preliminaries

A [*finite*] *labelled directed graph* over a set Σ of labels is defined as an ordered triple $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}), \ell_{\mathcal{G}})$, where $V(\mathcal{G})$ is a [*finite*] *set of vertices* of \mathcal{G} , $E(\mathcal{G})$ is a subset of $V(\mathcal{G}) \times V(\mathcal{G})$ called the *set of edges* of \mathcal{G} , and $\ell_{\mathcal{G}}$ is a function from $V(\mathcal{G})$ into Σ called the *labelling function* of \mathcal{G} . We drop the adjective ‘directed’ if there is no risk of confusion.

A *homomorphism of a labelled directed graph \mathcal{G} over Σ into a labelled directed graph \mathcal{G}' over Σ* is an ordered triple $(\mathcal{G}, h : V(\mathcal{G}) \rightarrow V(\mathcal{G}'), \mathcal{G}')$ such that h is a function from $V(\mathcal{G})$ into $V(\mathcal{G}')$ which satisfies the following conditions:

- (H₁) $(v, v') \in E(\mathcal{G})$ implies $(h(v), h(v')) \in E(\mathcal{G}')$ for all $v, v' \in V(\mathcal{G})$,
- (H₂) $\ell_{\mathcal{G}'}(h(v)) = \ell_{\mathcal{G}}(v)$ for every $v \in V(\mathcal{G})$.

If a triple $h = (\mathcal{G}, h : V(\mathcal{G}) \rightarrow V(\mathcal{G}'), \mathcal{G}')$ is a homomorphism of a labelled directed graph \mathcal{G} over Σ into a labelled directed graph \mathcal{G}' over Σ , we denote this triple by $h : \mathcal{G} \rightarrow \mathcal{G}'$, we write $\text{dom}(h)$ and $\text{cod}(h)$ for \mathcal{G} and \mathcal{G}' , respectively, according to category theory convention, and we write $h(v)$ for the value $h(v)$.

A homomorphism $h : \mathcal{G} \rightarrow \mathcal{G}'$ of labelled directed graphs over Σ is an *embedding of \mathcal{G} into \mathcal{G}'* , denoted by $h : \mathcal{G} \hookrightarrow \mathcal{G}'$, if the following condition holds:

- (E) $h(v) = h(v')$ implies $v = v'$ for all $v, v' \in V(\mathcal{G})$.

An embedding $h : \mathcal{G} \hookrightarrow \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ is an *inclusion of \mathcal{G} into \mathcal{G}'* , denoted by $h : \mathcal{G} \hookrightarrow \mathcal{G}'$, if the following holds:

- (I) $h(v) = v$ for every $v \in V(\mathcal{G})$.

We say that a labelled directed graph \mathcal{G} over Σ is a *labelled subgraph* of a labelled directed graph \mathcal{G}' over Σ if there exists an inclusion $h : \mathcal{G} \hookrightarrow \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ .

For an embedding $h : \mathcal{G} \hookrightarrow \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ we define the *image* of h , denoted by $\text{im}(h)$, to be a labelled directed graph $\widehat{\mathcal{G}}$ over Σ such that $V(\widehat{\mathcal{G}}) = \{h(v) \mid v \in V(\mathcal{G})\}$, $E(\widehat{\mathcal{G}}) = \{(h(v), h(v')) \mid (v, v') \in E(\mathcal{G})\}$, and the labelling function $\ell_{\widehat{\mathcal{G}}}$ of $\widehat{\mathcal{G}}$ is the restriction of the labelling function $\ell_{\mathcal{G}'}$ of $V(\mathcal{G}')$ to the set $V(\widehat{\mathcal{G}})$, i.e., $\ell_{\widehat{\mathcal{G}}}(v) = \ell_{\mathcal{G}'}(v)$ for every $v \in V(\widehat{\mathcal{G}})$.

A homomorphism $h : \mathcal{G} \rightarrow \mathcal{G}'$ of labelled directed graphs over Σ is an *isomorphism* of \mathcal{G} into \mathcal{G}' if there exists a homomorphism $h^{-1} : \mathcal{G}' \rightarrow \mathcal{G}$ of labelled directed graphs over Σ , called the *inverse* of h , such that the following conditions hold:

- (Iz₁) $h^{-1}(h(v)) = v$ for every $v \in V(\mathcal{G})$,
- (Iz₂) $h(h^{-1}(v)) = v$ for every $v \in V(\mathcal{G}')$.

We say that a labelled directed graph \mathcal{G} over Σ is *isomorphic* to a labelled directed graph \mathcal{G}' over Σ if there exists an isomorphism $h : \mathcal{G} \rightarrow \mathcal{G}'$ of labelled graphs $\mathcal{G}, \mathcal{G}'$ over Σ .

For an embedding $h : \mathcal{G} \hookrightarrow \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ we define a homomorphism $\dot{h} : \mathcal{G} \rightarrow \text{im}(h)$ by $\dot{h}(v) = h(v)$ for every $v \in V(\mathcal{G})$. This homomorphism \dot{h} is an isomorphism of \mathcal{G} into $\text{im}(h)$, called an *isomorphism deduced by h* .

For a labelled directed graph \mathcal{G} over Σ , the *identity homomorphism* (or simply, *identity of \mathcal{G}*), denoted by $\text{id}_{\mathcal{G}}$, is the homomorphism $h : \mathcal{G} \rightarrow \mathcal{G}$ such that $h(v) = v$ for every $v \in V(\mathcal{G})$.

We say that a labelled directed graph \mathcal{G} over Σ is an *isomorphically perfect* labelled directed graph over Σ if the identity homomorphism $\text{id}_{\mathcal{G}}$ is a unique isomorphism of labelled directed graph \mathcal{G} into \mathcal{G} .

Lemma A.1. *Let \mathcal{G} be an isomorphically perfect labelled directed graph over Σ and let $h : \mathcal{G} \rightarrow \mathcal{G}'$, $h' : \mathcal{G} \rightarrow \mathcal{G}'$ be two isomorphisms of labelled graphs $\mathcal{G}, \mathcal{G}'$ over Σ . Then $h = h'$.*

We say that a set or a class \mathcal{A} of labelled directed graphs over Σ is *skeletal* if for all labelled directed graphs $\mathcal{G}, \mathcal{G}'$ in \mathcal{A} if they are isomorphic, then $\mathcal{G} = \mathcal{G}'$.

A *gluing diagram* \mathcal{D} of labelled directed graphs over Σ is defined by:

- its *set \mathcal{I} of indexes* with a distinguished index $\Delta \in \mathcal{I}$, called the *center of \mathcal{D}* ,
- its *family \mathcal{G}_i ($i \in \mathcal{I}$) of labelled directed graphs over Σ* ,
- its *family gl_i ($i \in \mathcal{I} - \{\Delta\}$) of gluing conditions* which are sets of ordered pairs such that
 - (i) $\text{gl}_i \subseteq V(\mathcal{G}_{\Delta}) \times V(\mathcal{G}_i)$ for every $i \in \mathcal{I} - \{\Delta\}$,
 - (ii) $(v, v') \in \text{gl}_i$ implies $\ell_{\mathcal{G}_{\Delta}}(v) = \ell_{\mathcal{G}_i}(v')$ for all $v \in V(\mathcal{G}_{\Delta})$, $v' \in V(\mathcal{G}_i)$, and for every $i \in \mathcal{I} - \{\Delta\}$,
 - (iii) for every $i \in \mathcal{I} - \{\Delta\}$ if gl_i is non-empty, then there exists a bijection

$$b_i : L(\text{gl}_i) \rightarrow R(\text{gl}_i)$$

for $L(\text{gl}_i) = \{v \mid (v, v') \in \text{gl}_i \text{ for some } v'\}$ and $R(\text{gl}_i) = \{v' \mid (v, v') \in \text{gl}_i \text{ for some } v\}$ such that $\{(v, b_i(v)) \mid v \in L(\text{gl}_i)\} = \text{gl}_i$.

For a gluing diagram \mathcal{D} of labelled directed graphs over Σ we define a *cocone* of \mathcal{D} to be a family $h_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$) of homomorphisms of labelled directed graphs over Σ (here $\text{cod}(h_i) = \mathcal{G}$ for every $i \in \mathcal{I}$) such that

$$h_{\Delta}(v) = h_i(v')$$

for every pair $(v, v') \in \text{gl}_i$ and every $i \in \mathcal{I} - \{\Delta\}$.

A cocone $q_i : \mathcal{G}_i \rightarrow \tilde{\mathcal{G}}$ ($i \in \mathcal{I}$) of \mathcal{D} is called a *colimiting cocone of \mathcal{D}* if for every cocone $h_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$) of \mathcal{D} there exists a unique homomorphism $h : \tilde{\mathcal{G}} \rightarrow \mathcal{G}$ of labelled directed graphs $\tilde{\mathcal{G}}, \mathcal{G}$ over Σ such that $h(q_i(v)) = h_i(v)$ for every $v \in V(\mathcal{G}_i)$ and for every $i \in \mathcal{I}$. The labelled directed graph $\tilde{\mathcal{G}}$ is called a *colimit* of \mathcal{D} , the homomorphisms q_i ($i \in \mathcal{I}$) are called *canonical injections* and the unique homomorphism h is called the *mediating morphism* for $h_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$).

For a gluing diagram \mathcal{D} one constructs its colimit $\tilde{\mathcal{G}}$ in the following way:

- $V(\tilde{\mathcal{G}}) = \bigcup_{i \in \mathcal{I}} (V_i \times \{i\})$, where
 - $V_\Delta = V(\mathcal{G}_\Delta)$ for the center Δ of \mathcal{D} ,
 - $V_i = V(\mathcal{G}_i) - R(\text{gl}_i)$ for every $i \in \mathcal{I} - \{\Delta\}$,
- $E(\tilde{\mathcal{G}}) = \bigcup_{i \in \mathcal{I}} E_i$, where
 - $E_\Delta = \{((v, \Delta), (v', \Delta)) \mid (v, v') \in E(\mathcal{G}_\Delta)\}$ for the center Δ of \mathcal{D} ,
 - $E_i = \{((v, i), (v', i)) \mid (v, v') \in E(\mathcal{G}_i) \text{ and } \{v, v'\} \subseteq V_i\}$
 - $\cup \{((v, \Delta), (v', \Delta)) \mid (v, v'') \in \text{gl}_i, (v', v''') \in \text{gl}_i,$
 - and $(v'', v''') \in E(\mathcal{G}_i) \text{ for some } v'', v'''\}$
 - $\cup \{((v, \Delta), (v', i)) \mid v' \in V_i, (v, v'') \in \text{gl}_i \text{ and } (v'', v') \in E(\mathcal{G}_i) \text{ for some } v''\}$
 - $\cup \{((v, i), (v', \Delta)) \mid v \in V_i, (v', v'') \in \text{gl}_i \text{ and } (v, v'') \in E(\mathcal{G}_i) \text{ for some } v''\}$
 - for every $i \in \mathcal{I} - \{\Delta\}$,
- the labelling function $\ell_{\tilde{\mathcal{G}}}$ is defined by $\ell_{\tilde{\mathcal{G}}}((v, i)) = \ell_{\mathcal{G}_i}(v)$ for every $(v, i) \in V(\tilde{\mathcal{G}})$.

The definition of a colimiting cocone of a gluing diagram \mathcal{D} provides that any other colimit of \mathcal{D} is isomorphic to the colimit of \mathcal{D} constructed above. Hence one proves the following lemma.

Lemma A.2. *Let \mathcal{D} be a gluing diagram of labelled graphs over Σ . Then for every colimiting cocone $q_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$) of \mathcal{D} if $i' \neq i''$, then*

$$(V(\text{im}(q_{i'})) - V(\text{im}(q_\Delta))) \cap (V(\text{im}(q_{i''})) - V(\text{im}(q_\Delta))) = \emptyset$$

for all $i', i'' \in \mathcal{I} - \{\Delta\}$, where Δ is the center of \mathcal{D} and the elements of nonempty $V(\text{im}(q_i)) - V(\text{im}(q_\Delta))$ with $i \neq \Delta$ are ‘new’ elements and the elements of $V(\text{im}(q_\Delta))$ are ‘old’ elements.

A generalized gluing diagram \mathcal{D} of labelled directed graphs over Σ is defined by:

- its set \mathcal{I} of indexes with a distinguished index $\Delta \in \mathcal{I}$, called the center of \mathcal{D} ,
- its family \mathcal{G}_i ($i \in \mathcal{I}$) of labelled directed graphs over Σ ,
- its family gl_j^i ($(i, j) \in \mathcal{I} \times (\mathcal{I} - \{\Delta\})$ and $i \neq j$) of gluing conditions which are such that
 - the set $\mathcal{I}^\Delta = \mathcal{I}$ with families \mathcal{G}_i ($i \in \mathcal{I}$) and gl_i^Δ ($i \in \mathcal{I} - \{\Delta\}$) form a gluing diagram \mathcal{D}^Δ with Δ as the center of \mathcal{D}^Δ ,
 - for every $i \in \mathcal{I} - \{\Delta\}$ the set $\mathcal{I}^i = \mathcal{I} - \{\Delta\}$ with families \mathcal{G}_i ($i \in \mathcal{I} - \{\Delta\}$) and gl_j^i ($j \in \mathcal{I} - \{i, \Delta\}$) form a gluing diagram \mathcal{D}^i with i as the center for \mathcal{D}^i ,
 - the following conditions hold:
 - (G₁) $R(\text{gl}_i^\Delta) \cap L(\text{gl}_j^i) = \emptyset$ for all i, j with $\{i, j\} \subset \mathcal{I} - \{\Delta\}$ and $i \neq j$,
 - (G₂) $(\text{gl}_j^i)^{-1} = \text{gl}_i^j$ for all i, j with $\{i, j\} \subset \mathcal{I} - \{\Delta\}$ and $i \neq j$, where for $Q \subset A \times B$

$$(Q)^{-1} = \{(x, y) \in B \times A \mid (y, x) \in A \times B\}.$$

For a generalized gluing diagram \mathcal{D} of labelled directed graphs over Σ we define a *cocone* of \mathcal{D} to be a family $h_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$) of homomorphisms of labelled directed graphs over Σ (here $\text{cod}(h_i) = \mathcal{G}$ for every $i \in \mathcal{I}$) such that for every $i \in \mathcal{I}$ the sub-family $h_j : \mathcal{G}_j \rightarrow \mathcal{G}$ ($j \in \mathcal{I}^i$) is a cocone of the diagram \mathcal{D}^i .

For a generalized gluing diagram \mathcal{D} a *colimiting cocone* of \mathcal{D} , a *colimit* of \mathcal{D} , the *canonical injections*, and the *mediating morphism* are defined in the same way as for a gluing diagram, e.g. a cocone $q_i : \mathcal{G}_i \rightarrow \tilde{\mathcal{G}}$ ($i \in \mathcal{I}$) of \mathcal{D} is called a *colimiting cocone* of \mathcal{D} if for every cocone $h_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$) of \mathcal{D} there exists a unique homomorphism $h : \tilde{\mathcal{G}} \rightarrow \mathcal{G}$ of labelled directed graphs $\tilde{\mathcal{G}}, \mathcal{G}$ over Σ such that $h(q_i(v)) = h_i(v)$ for every $v \in V(\mathcal{G}_i)$ and for every $i \in \mathcal{I}$.

Lemma A.3. *Let \mathcal{D} be a generalized gluing diagram with finite set \mathcal{I} of its indexes and with center Δ , such that the following condition holds:*

$$(G_3) \text{ for all } i, i', j \in \mathcal{I} - \{\Delta\} \text{ if } i \neq i', \text{ then } L(\text{gl}_i^j) \cap L(\text{gl}_{i'}^j) = \emptyset.$$

Then one constructs a colimit of \mathcal{D} to be a labelled directed graph $\tilde{\mathcal{G}}$ which is determined by an arbitrary nonrepetitive sequence i_1, \dots, i_{n_0} of elements of $\mathcal{I} - \{\Delta\} = \{i_1, \dots, i_{n_0}\}$ and which is defined in the following way:

$$- V(\tilde{\mathcal{G}}) = \bigcup_{i \in \mathcal{I}} (V_i \times \{i\}), \text{ where } V_\Delta = V(\mathcal{G}_\Delta), V_{i_1} = V(\mathcal{G}_{i_1}) - R(\text{gl}_{i_1}^\Delta), \text{ for every } k \text{ with } 1 < k \leq n_0$$

$$V_{i_k} = V(\mathcal{G}_{i_k}) - \left(R(\text{gl}_{i_k}^\Delta) \cup \bigcup_{1 \leq m < k} L(\text{gl}_{i_m}^{i_k}) \right),$$

$$- E(\tilde{\mathcal{G}}) = \bigcup_{i \in \mathcal{I}} E_i, \text{ where } E_\Delta = \{((v, \Delta), (v', \Delta)) \mid (v, v') \in E(\mathcal{G}_\Delta)\},$$

for every $i \in \mathcal{I} - \{\Delta\}$

$E_i = E_i^1 \cup E_i^2 \cup E_i^3 \cup E_i^4$ for

$$E_i^1 = \{((v, i), (v', i)) \mid \{(v, i), (v', i)\} \subset V(\tilde{\mathcal{G}}) \text{ and } (v, v') \in E(\mathcal{G}_i)\},$$

$$E_i^2 = \{((v, k), (v', j)) \mid \{(v, k), (v', j)\} \subset V(\tilde{\mathcal{G}}), i \notin \{k, j\} \subset \mathcal{I},$$

$$(v, v'') \in \text{gl}_i^k, (v', v''') \in \text{gl}_i^j, \text{ and } (v'', v''') \in E(\mathcal{G}_i) \text{ for some } v'', v'''\},$$

$$E_i^3 = \{((v, i), (v', j)) \mid \{(v, i), (v', j)\} \subset V(\tilde{\mathcal{G}}), i \neq j \in \mathcal{I},$$

$$(v', v'') \in \text{gl}_i^j, \text{ and } (v, v'') \in E(\mathcal{G}_i) \text{ for some } v''\},$$

$$E_i^4 = \{((v, j), (v', i)) \mid \{(v, j), (v', i)\} \subset V(\tilde{\mathcal{G}}), i \neq j \in \mathcal{I},$$

$$(v, v'') \in \text{gl}_i^j, \text{ and } (v'', v') \in E(\mathcal{G}_i) \text{ for some } v''\},$$

— the labelling function $\ell_{\tilde{\mathcal{G}}}$ is defined by $\ell_{\tilde{\mathcal{G}}}((v, i)) = \ell_{\mathcal{G}_i}(v)$ for every $(v, i) \in V(\tilde{\mathcal{G}})$.

Proof. Since by (G_3) for all $i \in \mathcal{I} - \{\Delta\}$ and $v \in V(\mathcal{G}_i) - V_i$ there exists a unique ordered pair $(v^*, i^*) \in V(\tilde{\mathcal{G}})$ such that $(v^*, v) \in \text{gl}_i^{i^*}$, one defines the i -th component $q_i : \mathcal{G}_i \rightarrow \tilde{\mathcal{G}}$ ($i \in \mathcal{I} - \{\Delta\}$) of colimiting cocone by

$$q_i(v) = (v, i) \text{ if } v \in V_i, (v^*, i^*) \text{ otherwise.} \quad \square$$

Lemma A.4. *Let \mathcal{D} be a generalized gluing diagram with finite set \mathcal{I} of its indexes and with center Δ , such that the condition (G_3) holds and let $q_i : \mathcal{G}_i \rightarrow \mathcal{G}$ ($i \in \mathcal{I}$) be a colimiting cocone of \mathcal{D} . Then for every $H \subseteq \mathcal{I} - \{\Delta\}$ if*

$$\bigcap_{i \in H} (V(\text{im}(q_i)) - V(\text{im}(q_\Delta))) \neq \emptyset,$$

then H has at most two elements and if $H = \{i, i'\}$ with $i \neq i'$, then $\text{gl}_{i'}^i$ is nonempty.

Proof. The lemma is a consequence of Lemma A.3 and the fact that two different colimits of a generalized gluing diagram are always isomorphic labelled graphs.

For two directed graphs $G_1 = (V(G_1), E(G_1))$, $G_2 = (V(G_2), E(G_2))$, we define their *union* by

$$G_1 \cup G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2)).$$

We introduce the following new concepts.

By a *directed multi-hypergraph* we mean a structure \mathcal{G} given by its set $E(\mathcal{G})$ of hyperedges, its set $V(\mathcal{G})$ of vertices and the source and target mappings

$$s_{\mathcal{G}} : E(\mathcal{G}) \rightarrow \mathcal{P}(V(\mathcal{G})), \quad t_{\mathcal{G}} : E(\mathcal{G}) \rightarrow \mathcal{P}(V(\mathcal{G}))$$

such that $V(\mathcal{G})$ together with

$$\{(\mathcal{V}_1, \mathcal{V}_2) \mid s_{\mathcal{G}}(e) = \mathcal{V}_1 \text{ and } t_{\mathcal{G}}(e) = \mathcal{V}_2 \text{ for some } e \in E(\mathcal{G})\}$$

form a directed hypergraph as in [5], where $\mathcal{P}(X)$ denotes the set of all subsets of a set X .

We say that two directed multi-hypergraphs $\mathcal{G}, \mathcal{G}'$ are *isomorphic* if there exist two bijections $h : V(\mathcal{G}) \rightarrow V(\mathcal{G}')$, $h' : E(\mathcal{G}) \rightarrow E(\mathcal{G}')$ such that

$$s_{\mathcal{G}'}(h'(e)) = \{h(v) \mid v \in s_{\mathcal{G}}(e)\} \text{ and } t_{\mathcal{G}'}(h'(e)) = \{h(v) \mid v \in t_{\mathcal{G}}(e)\}$$

for all $e \in E(\mathcal{G})$.

Appendix B

We recall an idea of a Gandy–Păun–Rozenberg machine, briefly G–P–R machine, introduced in [14].

The core of a G–P–R machine is a finite set of rewriting rules for certain finite directed labelled graphs, where these graphs are instantaneous descriptions for the computation process realized by the machine.

The conflictless parallel (simultaneous) application of the rewriting rules of a G–P–R machine is realized in Gandy’s machine mode (according to Local Causation Principle, cf. [3]), where (local) maximality of “causal neighbourhoods” replaces (global) maximality of, e.g. conflictless set of evolution rules applied

simultaneously to a membrane structure which appears during the evolution process generated by a P system [17]. Therefore one can construct a Gandy's machine from a G-P-R machine in an immediate way, see [14].

For all unexplained terms and notation of category theory and graph theory we refer the reader to Appendix A.

Definition B.1. A G-P-R machine \mathcal{M} is determined by the following data:

- a finite set $\Sigma_{\mathcal{M}}$ of labels or symbols of \mathcal{M} ,
- a skeletal set $\mathcal{S}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over Σ , which are called *instantaneous descriptions* of \mathcal{M} ,
- a function $\mathcal{F}_{\mathcal{M}} : \mathcal{S}_{\mathcal{M}} \rightarrow \mathcal{S}_{\mathcal{M}}$ called the *transition function* of \mathcal{M} ,
- a function $\mathcal{R}_{\mathcal{M}} : \text{PREM}_{\mathcal{M}} \rightarrow \text{CONCL}_{\mathcal{M}}$ from a finite skeletal set $\text{PREM}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ onto a finite skeletal set $\text{CONCL}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ such that $\mathcal{R}_{\mathcal{M}}$ determines the set

$$\tilde{\mathcal{R}}_{\mathcal{M}} = \{P \vdash C \mid P \in \text{PREM}_{\mathcal{M}} \text{ and } C = \mathcal{R}_{\mathcal{M}}(P)\}$$

of *rewriting rules* of \mathcal{M} which are identified with ordered pairs $r = (P_r, C_r)$, where the graph $P_r \in \text{PREM}_{\mathcal{M}}$ is the *premise* of r and the graph $C_r = \mathcal{R}_{\mathcal{M}}(P_r)$ is the *conclusion* of r ,

- a subset $\mathcal{I}_{\mathcal{M}}$ of $\mathcal{S}_{\mathcal{M}}$ which is the set of *initial instantaneous descriptions* of \mathcal{M} .

The above data are subject of the following conditions:

- 1) $V(\mathcal{G}) \subseteq V(\mathcal{F}_{\mathcal{M}}(\mathcal{G}))$ for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$,
- 2) $V(\mathcal{G}) \subseteq V(\mathcal{R}_{\mathcal{M}}(\mathcal{G}))$ for every $\mathcal{G} \in \text{PREM}_{\mathcal{M}}$,
- 3) the rewriting rules of \mathcal{M} are *applicable* to $\mathcal{S}_{\mathcal{M}}$ which means that for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$ the set

$$\begin{aligned} \mathcal{P}\ell(\mathcal{G}) = \{ & h \mid h \text{ is an embedding of labelled graphs over } \Sigma \\ & \text{with } \text{dom}(h) \in \text{PREM}_{\mathcal{M}} \text{ and } \text{cod}(h) = \mathcal{G} \\ & \text{such that for every embedding } h' \text{ of labelled graphs over } \Sigma \\ & \text{with } \text{dom}(h') \in \text{PREM}_{\mathcal{M}} \text{ and } \text{cod}(h') = \mathcal{G} \\ & \text{if } \text{im}(h) \text{ is a labelled subgraph of } \text{im}(h'), \text{ then } h = h'\} \end{aligned}$$

of *maximal applications*⁴ h of the rules $\text{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\text{dom}(h))$ of \mathcal{M} in places $\text{im}(h)$ is such that the following conditions hold:

$$(i) \quad V(\mathcal{G}) = \bigcup_{h \in \mathcal{P}\ell(\mathcal{G})} V(\text{im}(h)), \quad E(\mathcal{G}) = \bigcup_{h \in \mathcal{P}\ell(\mathcal{G})} E(\text{im}(h)),$$

⁴ with respect to the relation of being a labelled subgraph which can be treated as a natural priority relation between the applications of the rewriting rules

- (ii) for all $h_1, h_2 \in \mathcal{P}\ell(\mathcal{G})$ the equation $\ell_{\mathcal{G}_{h_1}}(\dot{h}_1^{-1}(v)) = \ell_{\mathcal{G}_{h_2}}(\dot{h}_2^{-1}(v))$ holds for every $v \in V(\text{im}(h_1)) \cap V(\text{im}(h_2))$, where $\ell_{\mathcal{G}_{h_1}}, \ell_{\mathcal{G}_{h_2}}$ are the labelling functions of $\mathcal{G}_{h_1} = \mathcal{R}_{\mathcal{M}}(\text{dom}(h_1)), \mathcal{G}_{h_2} = \mathcal{R}_{\mathcal{M}}(\text{dom}(h_2))$, respectively, and $\dot{h}_1^{-1}, \dot{h}_2^{-1}$ are the inverses of isomorphisms induced by the embeddings h_1, h_2 , respectively.
- (iii) $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is a colimit of a gluing diagram $\mathcal{D}^{\mathcal{G}}$ constructed in the following way (the construction of $\mathcal{D}^{\mathcal{G}}$ is provided by (ii)):
- the set \mathcal{I} of indexes of $\mathcal{D}^{\mathcal{G}}$ is such that $\mathcal{I} = \mathcal{P}\ell(\mathcal{G}) \cup \{\Delta\}$, where $\Delta \notin \mathcal{P}\ell(\mathcal{G})$ is the center of $\mathcal{D}^{\mathcal{G}}$,
 - the family \mathcal{G}_i ($i \in \mathcal{I}$) of labelled graphs of $\mathcal{D}^{\mathcal{G}}$ is such that $\mathcal{G}_h = \mathcal{R}_{\mathcal{M}}(\text{dom}(h))$ for every $h \in \mathcal{P}\ell(\mathcal{G})$, and \mathcal{G}_{Δ} is such that $V(\mathcal{G}_{\Delta}) = V(\mathcal{G}), E(\mathcal{G}_{\Delta}) = \emptyset$, and the labelling function $\ell_{\mathcal{G}_{\Delta}}$ is such that provided by (ii)

$$\ell_{\mathcal{G}_{\Delta}}(v) = \ell_{\mathcal{G}_h}(\dot{h}^{-1}(v))$$

for every $v \in V(\text{im}(h))$ and every $h \in \mathcal{P}\ell(\mathcal{G})$, where \dot{h}^{-1} is the inverse of the isomorphism \dot{h} induced by the embedding h ,

- the gluing conditions gl_h ($h \in \mathcal{P}\ell(\mathcal{G})$) of $\mathcal{D}^{\mathcal{G}}$ are defined by

$$\text{gl}_h = \{(v, \dot{h}^{-1}(v)) \mid v \in V(\text{im}(h))\}$$

for every $h \in \mathcal{P}\ell(\mathcal{G})$, where \dot{h}^{-1} is the inverse of the isomorphism \dot{h} induced by embedding h ,

- (iv) the following equations hold:

$$V(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) = \bigcup_{i \in \mathcal{I}} V(\text{im}(q_i))$$

$$\text{and } E(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) = \bigcup_{i \in \mathcal{I}} E(\text{im}(q_i))$$

for the canonical injections $q_i : \mathcal{G}_i \rightarrow \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ ($i \in \mathcal{I}$) forming a colimiting cocone of the diagram $\mathcal{D}^{\mathcal{G}}$ defined in (iii),

- (v) the canonical injection $q_{\Delta} : \mathcal{G}_{\Delta} \rightarrow \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is an inclusion of labelled graphs, where Δ is the center of $\mathcal{D}^{\mathcal{G}}$ and q_{Δ} is Δ -th element of the colimiting cocone in (iv).

Thus $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is the result of simultaneous application of the rules $\text{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\text{dom}(h))$ in the places $\text{im}(h)$ for $h \in \mathcal{P}\ell(\mathcal{G})$, where one replaces simultaneously $\text{im}(h)$ by $\text{im}(q_h)$ in \mathcal{G} for $h \in \mathcal{P}\ell(\mathcal{G})$, respectively.

A finite sequence $(\mathcal{F}_{\mathcal{M}}^i(\mathcal{G}))_{i=0}^n$ is called a *finite computation of \mathcal{M}* , the number n is called the *time* of this computation, and $\mathcal{F}_{\mathcal{M}}^n(\mathcal{G})$ is called the *final instantaneous description* for this computation if

$$\mathcal{F}_{\mathcal{M}}^0(\mathcal{G}) = \mathcal{G} \in \mathcal{I}_{\mathcal{M}}, \quad \mathcal{F}_{\mathcal{M}}^{n-1}(\mathcal{G}) \neq \mathcal{F}_{\mathcal{M}}^n(\mathcal{G}), \quad \text{and } \mathcal{F}_{\mathcal{M}}(\mathcal{F}_{\mathcal{M}}^n(\mathcal{G})) = \mathcal{F}_{\mathcal{M}}^n(\mathcal{G}),$$

where $\mathcal{F}_{\mathcal{M}}^i(\mathcal{G})$ is defined inductively: $\mathcal{F}_{\mathcal{M}}^i(\mathcal{G}) = \mathcal{F}_{\mathcal{M}}(\mathcal{F}_{\mathcal{M}}^{i-1}(\mathcal{G}))$.

For a computation $(\mathcal{F}_{\mathcal{M}}^i(\mathcal{G}))_{i=0}^n$ its *space* is defined by

$$\text{space}(\mathcal{M}, \mathcal{G}) = \max\{\text{the number of elements of } V(\mathcal{F}_{\mathcal{M}}^i(\mathcal{G})) \mid 0 \leq i \leq n\}$$

for $\mathcal{G} \in \mathcal{I}_{\mathcal{M}}$, where intuitively $\text{space}(\mathcal{M}, \mathcal{G})$ is understood as the size of hardware measured by the number of indecomposable processors⁵ used in the computations.

We recall the following definition from [15].

Definition B.2. A *generalized G–P–R machine* \mathcal{M} is defined by the following data:

- the sets $\Sigma_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}}, \mathcal{I}_{\mathcal{M}}$ and the functions $\mathcal{R}_{\mathcal{M}} : \text{PREM}_{\mathcal{M}} \rightarrow \text{CONCL}_{\mathcal{M}}, \mathcal{F}_{\mathcal{M}} : \mathcal{S}_{\mathcal{M}} \rightarrow \mathcal{S}_{\mathcal{M}}$, where $\mathcal{S}_{\mathcal{M}}, \text{PREM}_{\mathcal{M}}, \text{CONCL}_{\mathcal{M}}$ are skeletal sets of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$, the sets $\Sigma_{\mathcal{M}}, \text{PREM}_{\mathcal{M}}, \text{CONCL}_{\mathcal{M}}$ are finite sets, the condition 2) holds for $\mathcal{R}_{\mathcal{M}}$, and $\mathcal{I}_{\mathcal{M}}$ is a subset of $\mathcal{S}_{\mathcal{M}}$;
- besides the function $\mathcal{R}_{\mathcal{M}}$ defining *rewriting rules* there is enclosed a new function $\mathcal{R}_{\mathcal{M}}^a : \text{PREM}_{\mathcal{M}}^a \rightarrow \text{CONCL}_{\mathcal{M}}^a$, where $\text{PREM}_{\mathcal{M}}^a, \text{CONCL}_{\mathcal{M}}^a$ are finite skeletal sets of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ and $\mathcal{R}_{\mathcal{M}}^a$ defines *auxiliary gluing rules* $P \overset{a}{\vdash} C$ ($P \in \text{PREM}_{\mathcal{M}}^a, C = \mathcal{R}_{\mathcal{M}}^a(P)$) for defining common parts of the boundaries of new compartments appearing in a step of an evolution process;
- the above data are subject of the following conditions:
 - A) for every $\mathcal{G} \in \text{PREM}_{\mathcal{M}}^a$ we have $V(\mathcal{G}) \subseteq V(\mathcal{R}_{\mathcal{M}}^a(\mathcal{G}))$, the set $\mathcal{P}\ell(\mathcal{G})$ defined as in 3) satisfies 3)(ii), and there exists a generalized gluing diagram $\mathcal{D}_{\langle \mathcal{G} \rangle}$, called *gluing pattern determined by \mathcal{G}* , such that
 - a₁) the set $\mathcal{I}_{\langle \mathcal{G} \rangle}$ of indexes of $\mathcal{D}_{\langle \mathcal{G} \rangle}$ is a set $\{\Delta\} \cup \dot{\mathcal{I}}_{\langle \mathcal{G} \rangle}$ with Δ being the center of $\mathcal{D}_{\langle \mathcal{G} \rangle}$, $\dot{\mathcal{I}}_{\langle \mathcal{G} \rangle} \subseteq \mathcal{P}\ell(\mathcal{G})$, and $\Delta \notin \dot{\mathcal{I}}_{\langle \mathcal{G} \rangle}$;
 - a₂) the family of graphs \mathcal{G}_i ($i \in \mathcal{I}_{\langle \mathcal{G} \rangle}$) of $\mathcal{D}_{\langle \mathcal{G} \rangle}$ is such that $V(\mathcal{G}_{\Delta}) = V(\mathcal{G})$, $E(\mathcal{G}_{\Delta}) = \emptyset$, and $\mathcal{G}_h = \mathcal{R}_{\mathcal{M}}(\text{dom}(h))$ for $h \in \dot{\mathcal{I}}_{\langle \mathcal{G} \rangle}$;
 - a₃) the gluing conditions gl_i^{Δ} ($i \in \dot{\mathcal{I}}_{\langle \mathcal{G} \rangle}$) are such that $\text{gl}_i^{\Delta} = \text{gl}_i$ for gl_i defined as in 3)(iii) for the gluing diagram $\mathcal{D}^{\mathcal{G}}$;
 - a₄) $\mathcal{R}_{\mathcal{M}}^a(\mathcal{G})$ is a colimit of $\mathcal{D}_{\langle \mathcal{G} \rangle}$ with gluing conditions gl_j^i ($\{i, j\} \subseteq \dot{\mathcal{I}}_{\langle \mathcal{G} \rangle}$ and $i \neq j$) such that they are unique together with $\dot{\mathcal{I}}_{\langle \mathcal{G} \rangle}$ to make $\mathcal{R}_{\mathcal{M}}^a(\mathcal{G})$ a colimit of $\mathcal{D}_{\langle \mathcal{G} \rangle}$;
 - B) for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$ the following conditions hold:
 - b₁) for $\mathcal{P}\ell^a(\mathcal{G})$ defined as in 3) with $\text{PREM}_{\mathcal{M}}$ replaced by $\text{PREM}_{\mathcal{M}}^a$ and for every $h \in \mathcal{P}\ell^a(\mathcal{G})$ and gluing pattern $\mathcal{D}_{\langle \text{dom}(h) \rangle}$ determined by $\text{dom}(h)$ the set $\text{SCP}_h = \{h \circ h' \mid h' \in \dot{\mathcal{I}}_{\langle \text{dom}(h) \rangle}\}$, called the *scope of gluing pattern $\mathcal{D}_{\langle \text{dom}(h) \rangle}$ in place h* , is a subset of $\mathcal{P}\ell(\mathcal{G})$ defined as in 3) for \mathcal{G} and $\text{PREM}_{\mathcal{M}}$, where \circ denotes the composition of homomorphisms of graphs;
 - b₂) the set $\mathcal{P}\ell(\mathcal{G})$ defined in 3) satisfies conditions 3)(i), (ii);
 - b₃) the graph $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is a colimit of a generalized gluing diagram $\mathcal{D}_{\mathcal{G}}$ such that

⁵ The indecomposable processors coincide with urelements appearing in those Gandy machines which represent G–P–R machines in [14].

- (β_1) the set \mathcal{I} of indexes of $\mathcal{D}_{\mathcal{G}}$ is the same as the set of indexes of $\mathcal{D}^{\mathcal{G}}$ given in 3)(iii), i.e. $\mathcal{I} = \mathcal{P}\ell(\mathcal{G}) \cup \{\Delta\}$,
- (β_2) the family of graphs \mathcal{G}_i ($i \in \mathcal{I}$) of $\mathcal{D}_{\mathcal{G}}$ is the same as of $\mathcal{D}^{\mathcal{G}}$ defined in 3)(iii),
- (β_3) the gluing condition gl_i^{Δ} is gl_i defined in 3)(iii) for every $i \in \mathcal{I} - \{\Delta\}$,
- (β_4) for all h_1, h_2 with $\{h_1, h_2\} \subseteq \mathcal{I} - \{\Delta\}$ and $h_1 \neq h_2$ if there exists $h \in \mathcal{P}\ell^a(\mathcal{G})$ for which $\{h_1, h_2\} \subseteq \text{SCP}_h$, then the gluing condition $\text{gl}_{h_2}^{h_1}$ of $\mathcal{D}_{\mathcal{G}}$ is the gluing condition $\text{gl}_{h_2}^{h_1}$ of the gluing pattern determined by $\text{dom}(h)$ for h'_1, h'_2 such that $h \circ h'_1 = h_1$ and $h \circ h'_2 = h_2$,
- (β_5) if there does not exist $h \in \mathcal{P}\ell^a(\mathcal{G})$ such that $\{h_1, h_2\} \subseteq \text{SCP}_h$ for h_1, h_2 as in (β_4), then the gluing condition $\text{gl}_{h_2}^{h_1}$ of $\mathcal{D}_{\mathcal{G}}$ is defined to be the empty set;
- b_4) the colimiting cocone $q_i : \mathcal{G}_i \rightarrow \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ ($i \in \mathcal{I}$) of $\mathcal{D}_{\mathcal{G}}$ is such that
 - (β_6) the conditions 3)(iv) and (v) hold with $\mathcal{D}^{\mathcal{G}}$ replaced by $\mathcal{D}_{\mathcal{G}}$,
 - (β_7) for every at least two element subset H of $\mathcal{I} - \{\Delta\}$ such that

$$\bigcap_{i \in H} (V(\text{im}(q_i)) - V(\text{im}(q_{\Delta}))) \neq \emptyset$$

there exists $h \in \mathcal{P}\ell^a(\mathcal{G})$ such that H is a subset of SCP_h of gluing pattern determined by $\text{dom}(h)$.

The gluing conditions gl_j^i of $\mathcal{D}_{\mathcal{G}}$ defined in (β_4), (β_5) determine common parts of the boundaries of new compartments appearing in a step of an evolution process.

Appendix C

Basing on [18] we present the iterated function systems whose attractors are Koch curve and Sierpiński gasket, respectively. These iterated function systems consist of the bijections from \mathbb{R}^2 onto \mathbb{R}^2 (\mathbb{R}^2 denotes the set of ordered pairs of real numbers) described in terms of matrices as follows:

— for Koch curve

$$\begin{aligned} f_1^{\text{Koch}}(\mathbf{x}) &= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \mathbf{x} && \text{scale by } 1/3 \\ f_2^{\text{Koch}}(\mathbf{x}) &= \begin{bmatrix} 1/6 & -\sqrt{3}/6 \\ \sqrt{3}/6 & 1/6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/3 \\ 0 \end{bmatrix} && \text{scale by } 1/3, \text{ rotate by } 60^\circ \\ f_3^{\text{Koch}}(\mathbf{x}) &= \begin{bmatrix} 1/6 & \sqrt{3}/6 \\ -\sqrt{3}/6 & 1/6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/2 \\ \sqrt{3}/6 \end{bmatrix} && \text{scale by } 1/3, \text{ rotate by } -60^\circ \\ f_4^{\text{Koch}}(\mathbf{x}) &= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 2/3 \\ 0 \end{bmatrix} && \text{scale by } 1/3 \end{aligned}$$

— for Sierpiński gasket

$$\begin{aligned}
 f_1^{\text{Sierp}}(\mathbf{x}) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \mathbf{x} && \text{scale by } 1/2 \\
 f_2^{\text{Sierp}}(\mathbf{x}) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} && \text{scale by } 1/2 \\
 f_3^{\text{Sierp}}(\mathbf{x}) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/4 \\ \sqrt{3}/4 \end{bmatrix} && \text{scale by } 1/2
 \end{aligned}$$

References

1. Bolognesi T., *Causal sets from simple models of computation*, Int. Journal of Unconventional Computing **6** (2010), pp. 489–524.
2. Diekert V., Gastin P., *From local to global temporal logic over Mazurkiewicz traces*, Theoretical Computer Science **356** (2006), pp. 126–135.
3. Gandy R., *Church's thesis and principles for mechanisms*, in: The Kleene Symposium, eds. J. Barwise et al., North-Holland, Amsterdam 1980, pp. 123–148.
4. Gacs P., Levin L. A., *Causal nets or what is a deterministic computation*, Information and Control **51** (1981), pp. 1–19.
5. Gallo, G., Longo, G., Pallottino, S., Nguyen, S., *Directed hypergraphs and applications*, Discrete Appl. Math. **42** (1993), pp. 177–201.
6. Goertzel B., Review of *A New Kind of Science*, <http://www.goertzel.org/dynapsyc/2002/WolframReview.htm>
7. Joosten J. J., Soler-Toscano F., Zenil H., *Fractal dimension of space-time diagrams and the runtime complexity of small Turing machines*, Electronic Proceedings of Theoretical Computer Science **128** (2013), pp. 29–30.
8. Joosten J. J., Soler-Toscano F., Zenil H., *Fractal dimension versus computational complexity*, arXiv: 1309.1779v2 [cs.CC], 24 Mar 2014.
9. Lima-de-Faria A., *Atomic basis of biological symmetry and periodicity*, BioSystems **43** (1997), pp. 115–135.
10. Lima-de-Faria A., *Evolution Without Selection. Form and Function by Autoevolution*, Elsevier, Amsterdam, 1988.
11. Lutz J. H., *The dimensions of individual strings and sequences*, Information and Computation **187** (2003), pp. 49–79.
12. Markopoulou F., *The internal description of a causal set: what the universe looks like from inside*, arXiv: gr-gc/9811053v2, 18 Nov 1999.
13. Nowotny T., Requardt M., *Dimension theory of graphs and networks*, J. Phys. A Math. Gen. **31** (1998), pp. 2447–2463.
14. Obtułowicz A., *Randomized Gandy-Păun-Rozenberg machines*, in: Membrane Computing, Lecture Notes in Computer Science 6501, Springer, Berlin, 2011, pp. 305–324.
15. Obtułowicz A., *Generalized Gandy-Păun-Rozenberg machines for tile systems and cellular automata*, In: Conference on Membrane Computing 2011, Lecture Notes in Computer Science 7184, Springer, Berlin, 2012, pp. 314–322.
16. Obtułowicz A., *In search of a structure of fractals by using membranes as hyperedges*, In: Conference on Membrane Computing 2013, Lecture Notes in Computer Science 8340, Springer, Berlin, 2014, pp. 301–307.

17. Păun G., Rozenberg G., Salomaa A., *The Oxford Handbook of Membrane Computing*, Oxford, 2009.
18. Riddle L., *Classic iterated function system, Sierpiński gasket*, <http://ecademy.agnesscott.edu/~lriddle/ifs/siertri/siertri.htm>
19. Rozenfeld H. D., Gallos L. K., Song Ch., Makse H. A., *Fractal and transfractal scale-free networks*, Encyclopedia of Complexity and System Science, Springer, 2009, pp. 3924–3943.
20. Strogatz S. H., *Nonlinear Dynamics and Chaos*, Perseus Books Publ., LLC, 1994.
21. Volchan S. B., *What is a random sequence*, Amer. Math. Monthly **109** (2002), pp. 46–63.
22. Weibel E. R., *Design of biological organisms and fractal geometry*, in: Fractals in Biology and Medicine, ed. T. F. Nonnenmacher et al., Springer, Basel, 1994, pp. 68–85.
23. Wolfram, S., *Random sequence generated by cellular automata*, Advances in Appl. Math. **7** (1986), pp. 123–169.
24. Wolfram, S., *A New Kind of Science*, Wolfram Media, 2002.

P System Computational Model as Framework for Hybrid (Membrane-Quantum) Computations*

Yurii Rogozhin, Artiom Alhazov, Lyudmila Burtseva, Svetlana Cojocaru,
Alexandru Colesnicov, and Ludmila Malahov

Institute of Mathematics and Computer Science
5 Academiei st., Chisinau, Republic of Moldova, MD-2028
artiom@math.md, luburtseva@gmail.com, Svetlana.Cojocaru@math.md,
acolesnicov@gmx.com, lmalahov@gmail.com

Abstract. This work presents a hybrid model of high performance computations, representing the P system framework with additional quantum functionalities. This model is supposed to take advantages of both biomolecular and quantum paradigms and to overcome some of their inherent limitations. We extend a recently proposed formal model of interface between a membrane system and quantum sub-systems. The problem of finding the longest common subsequence for a set of strings is exhibited as an example.

Key words: Models of Computation, Parallelism and Concurrency, Quantum Computing, Biomolecular Computing, P Systems

1 Introduction

This research concerns the capability of the P system computational model [4] to provide a framework for hybrid calculating, employing additional functionality from a quantum model.

The first hybrid computational model by joining membrane- and quantum-approaches was proposed in the paper of A. Leporati [2] where QUREM (Quantum Unit Rules and Energy assigned to Membranes) P systems were introduced. Keeping the main frame of P systems, QUREM P systems change objects and rules: objects are represented as pure states of a quantum system, and rules are quantum operators. The result is a hybrid computation device, a membrane system with quantum operations, but computation itself is provided only by quantum formalism.

We propose our edition of hybrid computation model that keeps the entire expressivity of P systems. Classical P system formalism serves as the framework providing computations by elements of other models as well as communications between elements of different nature.

* The authors acknowledge the project STCU 5384 awarded by the Scientific and Technology Center in Ukraine.

In [6], we introduced the first version of our hybrid computation model. There, two types of membranes coexist: classical membranes and quantum membranes, the latter containing a quantum device inside. During the further research this conception has been extended by adding capability to perform quantum computations to any membrane.

To formulate quantum functionality incorporated in the proposed hybrid model, we use the classical scheme of quantum device [7]. Particular methods of quantum computation, which form the quantum functionality of the hybrid model, are selected according to the requirements of each problem being solved.

Proposed hybrid computations have to support data passing from the P system (macro) level to quantum (micro) level and back. The communication process is heart of hybrid computation model because it provides access to a different computational structure rather than just cloning existing ones. The communication is implemented by P system objects corresponding to the basis states of each initial and resulting qubit of the quantum device. The appearance of such objects in the membrane starts the quantum computation.

In [6] one presented hybrid solutions of two problems: SAT and image retrieval. In [5], the graph isomorphism problem was approached in the hybrid framework.

In this paper we will demonstrate the process of hybrid computation on the problem of finding the longest common subsequence for a set of strings that was selected because of its hard computation class, and suitability to be decomposed to different computational levels.

2 Hybrid Computational Model

2.1 Membrane Level of the Hybrid Model

We use standard **membrane systems**, or **P systems**, which consist of a hierarchy of membranes. We do not restrict ourselves by one specific variant of P systems selecting it in relation to the solved problem. For instance, two hybrid systems were presented in [6], one based on transitional non-cooperative P systems with atomic inhibitors, and the other based on tissue P systems with symport/antiport. In the present paper, we use **P systems with active membranes** as our membrane framework.

The **investigated model** will additionally suppose that, in any membrane, the apparition of some specific objects (quantum data, or quantum triggers) starts a quantum calculation. The said data are available as initial state of the quantum registers. After its termination the quantum calculation produces another specific objects (quantum results) inside the membrane. From the P system point of view, the quantum calculation is a step, or several steps, of the membrane calculation.

During quantum calculations in a membrane, application of P system rules to this membrane should be prohibited. Some aspects of this need more research. For example, division of a working quantum membrane would contradict the no-cloning theorem.

2.2 P Systems with Active Membranes

Definition 1. A P system with active membranes (without non-elementary membrane division) of initial degree $d \geq 1$ is a tuple $\Pi = (O, H, \mu, w_1, \dots, w_d, R)$, where

- O is a non-empty finite alphabet of objects,
- H is a finite set of membrane labels;
- μ is a membrane structure (i.e., a rooted unordered tree, usually represented by a bracketed expression) consisting of d membranes labeled (not necessarily injectively) by elements of H ;
- w_i , for every $i \in H$ mentioned in μ , are strings over O , describing the initial multisets of objects placed in the d regions of μ ;
- R is a finite set of rules.

Each membrane possesses, besides its label and position in μ , another attribute called polarization, which can be one of elements of set E . For the purposes of the present paper, $E = \{0, 1, 2\}$.

The rules are of the following kinds:

- (a) $[a \rightarrow u]_h^e$, $a \in O$, $u \in O^*$, $h \in H$, $e \in E$.
Object evolution rules. They can be applied inside a membrane labeled h with polarization e 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 all objects in u).
- (b) $a[]_h^e \rightarrow [b]_h^{e'}$, $a, b \in O$, $h \in H$, $e, e' \in E$.
Send-in communication rules. They can be applied to a membrane labeled h with polarization e and such that the external region contains an occurrence of the object a ; the object a is sent into h becoming b and, simultaneously, the polarization of h is changed to e' .
- (c) $[a]_h^e \rightarrow []_h^{e'} b$, $a, b \in O$, $h \in H$, $e, e' \in E$.
Send-out communication rules. They can be applied to a membrane labeled h with polarization e and containing an occurrence of the object a ; the object a is sent out from h to the outside region becoming b and, simultaneously, the polarization of h is changed to e' .
- (d) $[a]_h^e \rightarrow b$, $a, b \in O$, $h \in H$, $e \in E$.
Dissolution rules. They can be applied to a membrane labeled h with polarization e and containing an occurrence of the object a ; the membrane h is dissolved and its contents is released in the surrounding region, simultaneously changing an occurrence of a into b .
- (e) $[a]_h^e \rightarrow [b]_h^{e'} [c]_h^{e''}$, $a, b, c \in O$, $h \in H$, $e, e', e'' \in E$.
Elementary division rules. They can be applied to a membrane labeled h with polarization e , containing an occurrence of the object a , but having no other membrane inside (an elementary membrane); the membrane is divided into two membranes with the same label h and polarizations e' and e'' ; the object a is replaced, respectively, by b and c , while the other objects are copied to both membranes.

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

Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane any number of evolution rules can be applied simultaneously). The application of rules is maximally parallel: no further rules can be applied to the idle objects and membranes. If not indicated otherwise, we assume the standard semantics of application of P system rules.

A computation halts when no rules are applicable at some step.

In this paper, we consider P systems with input. We include the input sub-alphabet $\Sigma \subseteq O$ and the input region i_0 in the tuple specifying the P system. The computation starts after some input multiset of objects from Σ is added to the input region. As the result of a halting computation, the contents of a dedicated region is usually taken; sometimes, only a specific sub-alphabet is considered, and sometimes multiple regions are considered.

2.3 Quantum Level of the Hybrid Model

We suppose a standard **quantum device** available for quantum calculations. The quantum device contains qubits organized in quantum registers. It works in three steps: non-quantum (classical) initialization of qubits when they are set in base states; quantum transformation when the qubits are not observed; non-quantum (classical) measurement that produces the observable result.

Several restrictions are imposed over the quantum device. After the initialization and after the measurement each qubit is in one of base states $|0\rangle$, or $|1\rangle$. During quantum calculation a qubit may be in the superposition of both states, or it may be entangled with other qubits. The initial data and the result are regarded as non-negative integers in binary notation. The quantum transformation is linear and reversible. The general rule is that arguments and results are kept in different quantum registers. Another general condition is that the ancillary qubits were not entangled with the argument and the result after the calculation.

The construction of a quantum computer shown in Fig. 1 guarantees this. Indices show the dimensionality: the device implements the calculation of integer function $f : [0, 2^N - 1] \rightarrow [0, 2^M - 1]$. We may omit the index when we do not focus on the dimensionality. V_f is a quantum implementation of function f , and $V_f^\dagger = V_f^{-1}$ is the inverse transformation. It uses R ancillary qubits $|w\rangle$. Register $|z\rangle$ is initialized by quantum data that appeared in the membrane and made the quantum calculation to start. In the hybrid scheme below, quantum registers are initialized through the input objects $I_{k,b}$, where $b = 0, 1$ is a binary value of the k^{th} qubit in $|z\rangle$.

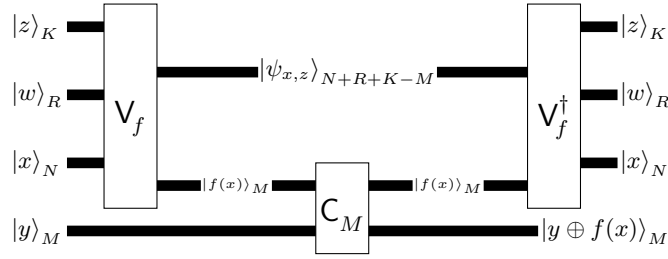


Fig. 1. Quantum calculation; initialization and measurement are not shown, indices denote the number of qubits in registers

2.4 Communication: Input/Output Signals and Triggering

We now focus on discussing the particularities of the hybrid model, namely, on the interaction between the biological sub-system and the quantum sub-system.

We now define the hybrid system as a tuple

$$\beta = (\Pi, T, T', H_Q, Q_N, Q_M, Inp, Outp, t, q_{h_1}, \dots, q_{h_m}).$$

Here, Π is a P system, and $H_Q = \{h_1, \dots, h_m\}$ is a subset of membrane labels in Π used for quantum calculations. T is a trigger and T' is the signal on obtaining the quantum result. Sub-systems q_{h_1}, \dots, q_{h_m} are the quantum sub-systems associated to the corresponding membranes from H_Q . The rest of the components of the tuple defining β specify the interaction between Π and q_{h_j} , $1 \leq j \leq m$.

For simplicity we assume that the running time quantum sub-systems of the same type is always the same. To keep this time general, we include a timing function $t : H_Q \rightarrow \mathbb{N}$: the quantum computation in a sub-system of type q_{h_j} takes $t(h_j)$ membrane steps. It is an open general question how to calculate the timing of quantum calculation with respect to the timing of membrane calculation. We could use as the first rough estimation that quantum calculation takes three steps of membrane calculation (initialization, quantum transformation, measurement).

The input size (in qubits) for quantum systems is given by $Q_N : H_Q \rightarrow \mathbb{N}$. The output size (in bits) for quantum systems is given by $Q_M : H_Q \rightarrow \mathbb{N}$.

We would like to define the behavior of β in all possible situations, so we introduce the trigger $T \in O$, where O is the alphabet of Π . The work of a quantum sub-system of type q_{h_j} starts whenever T appears inside the corresponding membrane. Note that we said q_{h_j} as a type of a quantum sub-system, because in general there may be multiple membranes with label h_j containing quantum sub-systems with the same functionality. The quantum state is initialized by objects from $Inp(h_j) = \{O_{k,h_j,b} \mid 1 \leq k \leq Q_N(h_j), b \in \{0,1\}\} \cup \{T\}$, so $Inp : H_Q \rightarrow 2^O$ is a function describing the input sub-alphabet for each type of quantum sub-system, the meaning of object $O_{k,h_j,b}$ being to initialize bit k of input by value b . We require that the set of rules satisfies the following condition: any object that may be sent into a membrane labeled h_j must be in $Inp(h_j)$.

If some bit k is not initialized, then the default value 0 is assumed. If multiple objects initialize bit k , say, $(O_{k,h_j,0})^s (O_{k,h_j,1})^t$, then the k -th qubit is set to

the state $\frac{1}{\sqrt{s^2+t^2}}(s|0\rangle + t|1\rangle)$. This technique is restricted: we can not produce entangled states. If some objects are sent to a quantum membrane without the trigger, we assume they wait there until the trigger arrives.

The output of quantum sub-systems is returned to the membrane system in the form of objects from $Outp(h_j) = \{R_{k,h_j,b} \mid 1 \leq k \leq Q_M(h_j), b \in \{0,1\}\} \cup \{T'\}$, the meaning of object $R_{k,h_j,b}$ being that the output bit k has value b . In case of one-bit output, we often denote it **yes** and **no**.

The result of a quantum sub-system is produced in the membrane together with object T' .

There are two possibilities to synchronize quantum and membrane levels. We can use a timing function and to wait for the quantum result by organizing the corresponding delay in membrane calculations, or we can wait for appearance of the resulting objects, or the trigger T' . For completeness, our model provides both possibilities. The topic needs further investigations.

The problem of some alternative evolution of the objects leading to the input into the quantum sub-system input can be always solved using a wrapper membrane (an extra membrane around the membrane used for quantum calculation), where the only rules applicable to the objects in it would be passing the input in, and passing the output out, if needed.

Another possible use of the wrapper membranes is to merge the input/output objects of the quantum sub-systems from objects of types $O_{k,h_j,b}$ and $R_{k,h_j,b}$ into objects of types $O_{k,b}$ and $R_{k,b}$, where the proper handling depending on h_j is done during their passage into/outside the wrapper membrane.

2.5 Initialization of the Quantum Device

Before the quantum calculation starts, each qubit is to be set in one of basis states $|0\rangle_1$, or $|1\rangle_1$. The measurement operation is embedded in the quantum device; it is used after calculation, so we can apply it to our qubits.

As we use the measurement before the calculation, the qubits collapse in the basis states. Measurement is irreversible, therefore it is a classical operation.

Now we are to set qubits in initial states. For example, if we want to set them in the state $|0\rangle$, we are to check the state of each qubits and invert $|1\rangle$. This is not a quantum transformation.

Usually, all qubits are initially set to $|0\rangle_1$. (Several quantum algorithms use different initial values though.) In our case, we will use non-quantum tools to prepare the state $|x\rangle|y\rangle|z\rangle|w\rangle = |0\rangle_N|0\rangle_M|z_0\rangle_K|0\rangle_R$. Here z_0 is the number that entered the quantum membrane and initiated the process. Classical (non-quantum) initialization ends here.

We provide qubits of our quantum device with the possibility to be initialized in some non-base states $\frac{1}{\sqrt{s^2+t^2}}(s|0\rangle + t|1\rangle)$. This is made by an appropriate one-qubit quantum transformation that should be performed before any other calculations.

For example, many quantum algorithms suppose that all qubits from register $|x\rangle$ are at once transformed by one-qubit Hadamard transformation

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

If $|x\rangle$ was set to $|0\rangle$ then each qubit of $|x\rangle$ is set in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. As the result, register $|x\rangle$ gets the state

$$|x\rangle = \frac{1}{2^{N/2}} \sum_{0 \leq i < 2^N} |i\rangle,$$

that corresponds to a uniform superposition of all possible values of argument x . Application of the transformation V_f will then produce all possible values of $f(x)$ (quantum parallelism).

Configuration of quantum device depends on parameters of a specific problem. Implementation of such dependence is sometimes called compilation. It is possible to delegate all such operation to membrane level, or to include them inside quantum device. They can be performed even at preparatory stage for the membrane level. Another variant is to consider these operations as an intermediate level of the system between membrane and quantum levels. The initialization of quantum device will also be a part of this level. It is logical as the initialization should be performed at the appearance of trigger T . We aim these problems for further research.

3 Finding the Longest Common Subsequence

3.1 Problem Formulation and Approach

The problem is given by strings $u_1, \dots, u_n \in V^*$, $V = \{a_1, \dots, a_k\}$. We assume without restricting generality that the first string is not longer than any other. We should find a string u_0 with the following properties:

1. u_0 is a subsequence of each u_i , $1 \leq i \leq n$;
2. u_0 is the longest string satisfying the previous property.

We consider the following way of solving the problem. Membrane division is used to consider the possible subsequences of the first string. The quantum sub-systems are used to verify if the candidate strings are subsequences of all other input strings. Delay is organized in order for the candidate strings to be considered in the longest-to-shortest order. If a solution is found in some membrane, then a signal goes to the skin, is replicated and sent into the membranes verifying the candidates to stop the process. The delay was chosen as a compromise in order not to slow down the overall system too much, while preventing most of unnecessary computations for shorter strings after a solution is found.

3.2 P System

In the construction of our membrane framework, we use numbers n and $n_i = |u_i|$, $1 \leq i \leq n$. The strings are represented by symbols from $\Sigma = \{a_{j,l,0,0} \mid 1 \leq j \leq n_1, 1 \leq l \leq k\} \cup \{b_{i,j,l,0} \mid 2 \leq i \leq n, 1 \leq j \leq n_i, 1 \leq l \leq k\}$ as follows: the string representing the input multiset is

$$w_{in} = \prod_{1 \leq j \leq n_1} (a_{j,l,0,0} \mid u_1[j] = a_l) \prod_{2 \leq i \leq n, 1 \leq j \leq n_i} (b_{i,j,l,0} \mid u_i[j] = a_l).$$

Let $K = \lceil \log_2 k \rceil$ be the number of bits needed to encode one symbol of V , and let $N = \lceil \log_2 (n_1 + 1) \rceil$ be the number of bits needed to represent a number between 0 and n_1 , inclusive.

We construct the following **hybrid** system.

$$\begin{aligned} \beta &= (\Pi, T, T', H_Q = \{2\}, Q_N, Q_M, Inp, Outp, t, q_2), \text{ where} \\ Q_N(2) &= N + K \sum_{j=1}^n n_j, \quad Q_M(2) = 1, \quad Outp(2) = \{\text{yes, no, } T'\}, \quad t(2) = 3, \\ Inp(2) &= \{T\} \cup \{I_{k,b} \mid 0 \leq k \leq N + K \sum_{t=1}^n n_t, 0 \leq b \leq 1\}, \\ q_2 &\text{ is a quantum system checking whether the first string} \\ &\text{is a subsequence of other strings, and} \\ \Pi &= (O, \Sigma, \mu = [[\]_2^0 [\]_3^0]_1^0, w_1, w_2, w_3, R, 2) \end{aligned}$$

is a **P system** with active membranes, where Σ is defined above, and

$$\begin{aligned} O &= \{d_j, s_j \mid 0 \leq j \leq n_1\} \cup \{\text{yes, yes}', \text{no}, o, p'_1, p'_0, T'\} \cup Inp(2) \\ &\cup \{c_{j,t} \mid 0 \leq j \leq n_1 + 1, 0 \leq t \leq j\} \cup \{c_t, c'_t \mid 0 \leq t \leq n_1\} \cup \{p_t \mid -1 \leq t \leq 5n_1\} \\ &\cup \{a_{j,l,s,t} \mid 1 \leq j \leq n_1, 1 \leq l \leq k, 0 \leq s \leq j, 0 \leq t \leq s\} \\ &\cup \{b_{i,j,l,s}, e_{j,l} \mid 1 \leq i \leq n, 1 \leq j \leq n_i, 1 \leq l \leq k, 1 \leq s \leq n_1 + 1\}, \\ w_1 &= \lambda, \quad w_2 = d_0 c_0, \quad w_3 = s_0, \end{aligned}$$

and the set R is the union of the following rule groups (presented together with their explanations): generation, counting, building subsequences, input for the quantum sub-systems, and stopping.

Generation

$$G_1. [d_j]_2^e \rightarrow [d_{j+1}]_2^0 [d_{j+1}]_2^1, \quad 0 \leq j \leq n_1 - 1, \quad 0 \leq e \leq 1.$$

Membrane 2 is divided n_1 times, each division representing the choice whether the subsequent symbol of the first string is selected for the candidate common subsequence.

$$G_2. [d_{n_1}]_2^e \rightarrow []_2^0 o, \quad 0 \leq e \leq 1.$$

Then the polarization is set to 0. Object o is not used in the system, it is only given because it is required by the types of the rules of P systems with active membranes.

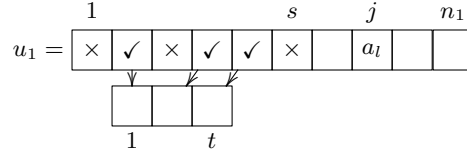


Fig. 2. Illustration of indices j, l, s, t appearing in objects $a_{j,l,s,t}$: string u_1 has symbol a_l in position j ; position s is being considered, and t symbols have been chosen for the candidate subsequence. Indices j, l, s have similar meaning for symbols $b_{i,j,l,s}$, except j refers to a position in u_i , while s refers to a position in u_1 .

Counting

$$C_1. [c_{j,t} \rightarrow c_{j+1,t+e}]_2^e, 0 \leq j \leq n_1, 0 \leq t \leq j, 0 \leq e \leq 1.$$

Subscript t counts the number of selected (by setting the polarization to 1) symbols, out of j symbols of the first string considered.

$$C_2. [c_{n_1+1,t} \rightarrow c_t p_{5(n_1-t)}]_2^0, 0 \leq t \leq n_1.$$

Remember the size of a chosen subsequence and initialize the delay counter.

$$C_3. [p_t \rightarrow p_{t-1}]_2^0, 1 \leq t \leq 5n_1.$$

Delay the computation for the number of steps equal to 5 times the number of not selected symbols. In this way, the candidate strings are considered the longest to the shortest, until a common subsequence is found. These rules can be disabled if the polarization is switched to 2.

$$C_4. [p_0 \rightarrow p_{-1} p'_1]_2^0.$$

$$C_5. [p_{-1}]_2^0 \rightarrow []_2^1 o.$$

Set the polarization to 1. In the next step, the input will be prepared for the quantum sub-system.

$$C_6. [p'_1 \rightarrow p'_0]_2^0.$$

$$C_7. [p'_0]_2^1 \rightarrow []_2^0 o.$$

The next step after having set the polarization to 1, it is again reset to 0 (simultaneously with initializing the quantum sub-system).

Building subsequences

$$B_1. [a_{j,l,s,t} \rightarrow a_{j,l,s+1,t+e}]_2^e, 1 \leq j \leq n_1, 1 \leq l \leq k, 0 \leq s \leq j-1, 0 \leq t \leq s, 0 \leq e \leq 1.$$

In the group $a_{j,l,s,t}$ of objects, the third subscript ranges over the positions of the first string, while the last subscript keeps track of the number of symbols chosen.

$$B_2. [a_{j,l,j,t} \rightarrow b_{1,t+1,l,j+1}]_2^1, 1 \leq j \leq n_1, 1 \leq l \leq k, 0 \leq t \leq s.$$

The j -th symbol of the first string is chosen as the t -th symbol of the candidate subsequence.

$$B_3. [a_{j,l,j,t} \rightarrow \lambda]_2^0, 1 \leq j \leq n_1, 1 \leq l \leq k, 0 \leq t \leq s.$$

The symbol is not chosen; the object representing it is erased.

$$B_4. [b_{i,j,l,s} \rightarrow b_{i,j,l,s+1}]_2^e, 1 \leq i \leq n, 1 \leq j \leq n_i, 1 \leq l \leq k, 1 \leq s \leq n_1.$$

Wait until the rest of the subsequence is chosen. These rules also apply to the input representing the other strings, synchronizing them.

Input for the quantum sub-systems

$I_1. [b_{i,j,l,n_1+1} \rightarrow f(i,j,l)]_2^1, 2 \leq i \leq n, 1 \leq j \leq n_i, 1 \leq l \leq k$, where

$$f(i,j,l) = I_{K(j-1+\sum_{t=2}^{i-1} n_t)+1,g(l,0)} \cdots I_{K(j-1+\sum_{t=2}^{i-1} n_t)+K,g(l,K-1)}, 2 \leq i \leq n,$$

$$f(1,j,l) = e_{j,l} I_{K(j-1+\sum_{t=2}^n n_t)+1,g(l,0)} \cdots I_{K(j-1+\sum_{t=2}^n n_t)+K,g(l,K-1)},$$

where $g(l,b) = \lfloor (l-1)/2^b \rfloor \bmod 2$ is the b -th bit of binary representation of number $l-1$.

$I_2. [c_t \rightarrow c'_t T f(1,t+1,0) \cdots f(1,n_1,0) F(t)]_2^1$ where

$$F(t) = I_{K(\sum_{t=1}^n n_t),g(t+1,0)} \cdots I_{K(\sum_{t=1}^n n_t)+N-1,g(t+1,N-1)}.$$

The total number of input objects passed to a quantum sub-system (besides the trigger T) is $N + K \sum_{t=1}^n n_t$.

Stopping

$S_1. [s_j \rightarrow s_{j+1} s_{j+1}]_3^0, 0 \leq j \leq n_1 - 1$.

2^{n_1} copies of object s_{n_1} are produced.

$S_2. [\text{yes} \rightarrow \text{yes}']_2^0$.

We make sure that the computation is not stopped (a longer string not found yet). We do not use the trigger T' produced together with **yes**.

$S_3. [\text{yes}']_2^0 \rightarrow []_2^1 \text{yes}$.

The polarization is set to 1.

$S_4. [c'_t]_2^1 \rightarrow []_2^1 c_t$.

The object is sent out storing the length of the found common string.

$S_5. c_t []_3^0 \rightarrow [c_t]_3^1$.

One object c_t enters membrane labeled 3, switching its polarization to make sure this is done once.

$S_6. [c_t]_3^1 \rightarrow c'_t$.

The object is primed again, dissolving membrane labeled 3, releasing objects s_{n_1} into the skin.

$S_7. s_{n_1} []_2^0 \rightarrow [o]_2^2$.

Objects s_{n_1} enter all membranes labeled 2, with polarization 0, stopping the computations there by setting the polarization to 2.

$S_8. [c'_t]_1^0 \rightarrow []_1^1 c_t$.

Object c_t is sent out to the environment, representing the length of the longest common subsequence.

The system described above solves the longest common subsequence problem. The length of this subsequence is sent out, stored in one object, while all longest subsequences (there may be a large number of them) are represented in the elementary membranes with polarization 1.

It is worth noticing that, similarly to the approach used in [6], the construction is made in such a way that all input data for the quantum sub-system is produced simultaneously and all qubits are initialized, so the construction does not require a trigger to synchronize the input; we have included it in rule I_2 just to comply with the presented model.

3.3 Quantum Calculation

The quantum calculation needed to unveil if one string is a subsequence of other(s) is thoroughly described in [3]. The algorithm is a modification of well-known Grover search algorithm. It uses the techniques originally introduced by Grover: a query operator that marks the state encoding the element being searched by changing its phase, followed by an amplitude amplification of the marked state. The state can be detected with non negligible probability by iterating this process several times. We do not focus on details in this paper.

4 Conclusion

This work continues the presentation of hybrid computational model that combines in the framework of membrane computation elements of others (for today, only quantum) approaches. In current research, the detailing of P system framework construction and functionality is provided, and interaction of membrane and quantum levels are described.

To obtain true hybrid computation the proposed hybrid model has to provide mutual accepting of input/output by computation models of different nature and even different (macro/micro) levels. We presented, in the current paper, the communication scheme in detail. The technique includes both P system and quantum tools. P system formalism as the framework of hybrid model provides converting multisets to quantum registers contents and back procedure. Quantum computation ends with measurement, which prepare the registers content for uploading back in P system.

Since the main reason of proposing of hybrid computational model was practical needs of several domains delivering hard tasks, we develop our model independently with its application to solution of selected set of such type problems. One of them was used as illustrative example in this work.

Basing on implemented solutions of several problems, we extract the common feature that makes hybrid computation possible. Solving tasks of this type mostly supposes generating of candidates lists that then will be used for comparison to pattern applying Grover search in the quantum device. In our further research, we plan to generalize these steps of solution both for P system and quantum devices.

The distribution of work between levels of hybrid system depends on the problem to be solved and may vary. In this paper we mostly use membrane replication to overpass the problem complexity while in [6] the power of quantum formalism was exploited. It would be interesting to combine both sources of

computational power to aim, for example, the Σ_2^P computational complexity class.

References

1. A. Alhazov, L. Burtseva, S. Cojocaru, A. Colesnicov, L. Malahov: An Approach to Implementation of Hybrid Computational Paradigm. *The Third Conference of Mathematical Society of the Republic of Moldova*, IMCS-50, 2014, accepted.
2. A. Leporati: P Systems with a Quantum-like Behavior: Background, Definition, and Computational Power. In: *Lecture Notes in Computer Science* **4860**, 2007, 32–53.
3. P. Mateus, Y. Omar: Quantum Pattern Matching, [arXiv:quant-ph/0508237v1](https://arxiv.org/abs/quant-ph/0508237v1)
4. Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
5. Yu. Rogozhin, A. Alhazov, L. Burtseva, S. Cojocaru, A. Colesnicov, L. Malahov: Hybrid (Membrane-Quantum) Model of High Performance Computations in Solving Problems of Computer Algebra Domain, in preparation.
6. Yu. Rogozhin, A. Alhazov, L. Burtseva, S. Cojocaru, A. Colesnicov, L. Malahov: Solving Problems in Various Domains by Hybrid Models of High Performance Computations. *Computer Science Journal of Moldova* **22**, 1(64), 2014, 3–20.
7. C.P. Williams: *Explorations in Quantum Computing*. Springer, 2008.

Expressing Workflow and Workflow Enactment using P Systems

Rohit Verma, Tanveer Ahmed, and Abhishek Srivastava

Department of Computer Science and Engineering
Indian Institute of Technology Indore, India
Email: {phd12110101, phd12120101, asrivastava}@iiti.ac.in

Abstract. Workflow can be viewed as an abstract term for purposeful collection of smaller subprocesses (work-items) for solving a bigger scientific or business problem. The recent advances in technology has enabled a decentralized and elastic execution of these work-items on distributed computing devices. We investigate elastic, decentralize, parallel and distributed methods of workflow enactment. We present a novel formal notation of the modern workflows. The main result of this research is a generic notation of workflows and their enactment, utilizing formal definitions of P systems. This research shows how complex modern workflows can be represented by using simpler membrane structures and evolution rules.

Keywords: Membrane computing; P systems; Workflow; Elastic computing

1 Introduction

Today various businesses and communities have designed several products and services that can be used together in collaboration to achieve some common task. Workflow technology makes collaboration of these distributed services flexible and efficient [8]. Service-oriented computing paradigm has made these loosely coupled, distributed, decentralized services to work in parallel. The multitude of these services by various business and scientific community has lead to the new concept of *Elastic Computing* in services and workflow execution. Further, the autonomy of these services have enabled decentralized realization of workflow, thus removing several bottlenecks such as scalability, reliability and fault tolerance (involved due to single point of failure of centralized execution management of the workflow).

Workflow is an abstract term used to describe the tasks or steps required to execute in the move to achieve a common goal. The execution of these tasks or steps (also known as work-items) are distributed and decentralized in nature, thus involves flow of information among various executional resources offered by organizations or people. While in execution, the computing resource requirement by the workflows might change due to unplanned requests, user interventions, change in execution environments. That requires a flexible and elastic resource

provisioning. Recent advancements in the technology [5, 9] has enabled elastic computing resource provisioning in workflows.

Inspired by this, we take a closer look at the decentralized and elastic business or scientific workflows. We regard this workflow as the synchronous collection of independent work-items collaborating to achieve a common goal. These individual work-items are the part of the business or scientific processes which are automated by the autonomous services provided by them. In elastic enactment of workflows, extra computing resources can be provisioned or deprovisioned as per the need and makes the execution resources elastic at run time. In a decentralized environment, each service executing a work-item is autonomous and is responsible for this elastic provisioning. We are using membrane systems (also known as P systems) [11] [12] to provide a formal notation of such workflows and their enactment. P system is the best suitable approach for providing formal notation for the enactment of these decentralized, distributed, autonomous and parallel workflows. Membrane system is a new class of distributed and parallel computing devices, inspired from the cell-structures introduced by Păun (2000). To the best of our knowledge, we are first to explore the possibilities of workflow enactment using membrane systems.

Instead of solving some particular problem, we propose a generic membrane system based framework for modeling and providing formal notation for the entire workflow execution, with special focus on the autonomous and elastic provisioning of computing resources. We propose to use Membrane Computing for providing a formal notation and generic framework for decentralized and elastic workflow enactment. Membrane computing paradigm is based on P System or Membrane systems and is motivated by the way nature computes at the cellular level. The fundamental features that are used in computing model are: Membrane structure, Evolutionary Rules and Objects. The workflow is represented as a main membrane which is a finite cell-structure consisting of other cell membranes (work-items), also called the *skin*. Membranes are the regions representing the *process-steps* or *work-items*. These membranes determine the regions where the evolutionary rules can be placed. These evolutionary rules are unique to each membrane (work-item) and are the set of functionalities which a work-item would perform in the move to execute the workflow. Any workflow execution would start from an initial configuration and would be executed successfully when no further rules can be applied. The term P system, membrane system and membrane computing will be used interchangeably in this paper.

The main contributions of this paper are: (1) A novel formal notation for decentralized and distributed workflow enactment, with autonomous and elastic provisioning of computing resources. (2) A generic notation for various workflows ranging from business workflows (control oriented), scientific workflows (resource intensive) to mobile workflows.

The rest of this paper is organised as follows. In Section 2 we recall the definition of membrane computing and formal P system notation, together with our proposed formal workflow definition. In Section 3 membrane oriented workflow definition is discussed. Furthermore, we discussed about the formal notation of

workflow from membrane perspective, solution of workflow pattern by P systems and novel features of proposed P system. In Section 4, we presented related work in the field of workflow. Finally, conclusion and future scope of the research is discussed in Section 5

2 Background

2.1 Membrane Computing Paradigm

Membrane computing paradigm is a new class of distributed and parallel computing devices introduced by Gheorghe Păun (2000) [11] [12]. Membrane computing is based on the membrane systems (or P systems) and is motivated by the way nature computes at the cellular level. P systems have been evolved inspired by the fact that biological membranes have strong relevance to the computing devices.

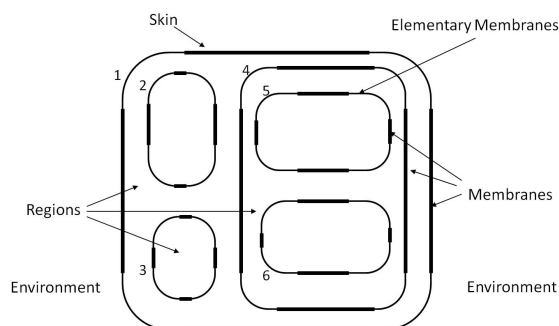


Fig. 1. Membrane Structure

A membrane structure is graphically represented by a Venn diagram as shown in Figure 1 [12], where a membrane may contain other membranes. The membrane structure is a hierarchical arrangement of membranes that are embedded in the *skin membrane* separating the inner membranes from external *environment*. *Elementary Membranes* are the membranes that do not have any other membrane inside. The membranes and the regions delimited by them are labeled with positive integers to address them distinctly. Furthermore, each region contains a *multi-set of objects* and a *set of evolution rules*.

Formal Definition of P system: A membrane structure (as shown in Figure 1) can be represented by a string of matching parenthesis as in:

$$[1[2]2[3]3[4[5]5[6]6]4]1$$

Definition 1: The formal basic (there are multiple definitions) definition of Membrane system (Π) according to Gheorghe Păun [11], is as follows:

$$\Pi = (O, T, C, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m, i_0)$$

where:

- i) O is the set of all objects.
- ii) T is the output alphabet and $T \subseteq O$.
- iii) C is set of catalysts and $C \cap T = \emptyset$.
- iv) μ is the membrane structure consisting of m membranes.
- v) w_1, w_2, \dots, w_m are the multisets of objects over O with the regions $1, 2, \dots, m$ of μ .
- vi) R_1, R_2, \dots, R_m are the evolutionary rules over O for each of the m membranes.
- vii) $i_0, m \in i_0$ represent the label of the output region.

Membrane Operations:

Before proceeding any further, we must outline the specialized operations [2] of the membrane computing paradigm. These operations are mandatory in the move to enact the workflow. Few of these operations are:

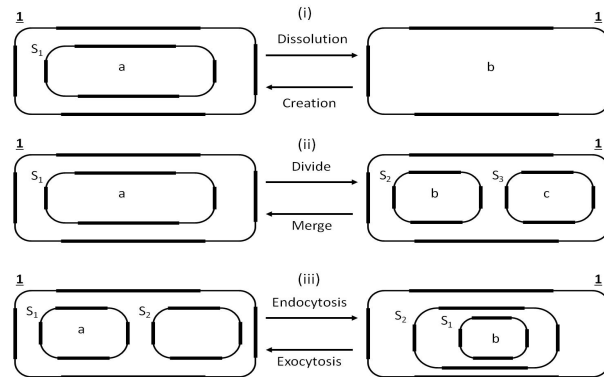


Fig. 2. Special Membrane Operations

Dissolution: Membrane dissolution operation indicates the realization of an activity by a membrane. The rule is depicted in (i) part of Figure 2. This is exemplified in evolution rule (1). In contrast to the dissolution rule, a creation rule signals the creation of a new membrane, that is depicted in (2).

$$[1[s_1 a]_{s_1}]_1 \rightarrow [1 b]_1 \delta \tag{1}$$

$$[1 b]_1 \rightarrow [1[s_1 a]_{s_1}]_1 \tag{2}$$

Division and Merge: The division rule of P systems, is also applied towards workflow enactment. This rule is particularly useful in cases where elastic execution of a ‘*process-step*’ takes precedence. This rule depicts the forking of a membrane (work-item) into multiple membranes (work-items) carrying the same dataset and evolution rules (as of parent membrane). Part (ii) of the Figure 2 and the rule (3) states the division of membrane and reverse of division operation states the merging of two membrane.

$$[1[s_1 a]s_1]_1 \rightarrow [1[s_2 b]s_2[s_3 c]s_3]_1 \quad (3)$$

Endocytosis: Endocytosis is a concept that is important to dynamically update the execution order of a workflow. The operation is shown in the Figure 2. This operation is depicted via the evolution rule (4):

$$[1[s_1 a]s_1[s_2]s_2]_1 \rightarrow [1[s_2[s_1 b]s_1]s_2]_1 \quad (4)$$

2.2 Workflow Definition

Workflow is the computerised facilitation or automation of a business process, in whole or part [7]. Workflow is often associated with business processes of an organization. However, workflow is also useful in other fields such as scientific computations. In scientific workflows, various computational tasks are processed in a specific order to solve a scientific problem. To proceed with the workflow execution, there are several constructs available in literature. In broad sense, workflow specification may have different perspectives [13]:

- *Control flow perspective* describes the flow of execution order among different tasks and work-items of a workflow. The control flow can be specified following certain basic patterns, such as: Sequence, Parallel Split, Synchronization (AND-join), Exclusive Choice (Decision), Merge.
- *Data flow perspective* deals with the flow of processing data needed to execute a workflow. This may include a business document or an object supplied to the workflow, local variables generated at the time of execution of individual work-items.
- *Resource flow perspective* provides the necessary infrastructure requirements needed for an efficient workflow execution. The requirement may range from human provided services to machine based physical resources.

Furthermore these workflow specifications can be sub-categorized into: *Abstract Workflow* and *Concrete Workflow* [3]. An abstract workflow specifies the solution of the problem along with the input data, but without containing any means for actual execution. In contrast, a concrete workflow specify the mapping between physical resources, responsible for executing (such as services provided by machines or human) the abstract work-items.

Definition 2: The workflow can be formally specified as:

$$W = (F, D, T, R)$$

where:

- i) F is the set of functions or work-items in the workflow.
- ii) D is the working dataset, where data $d_{i,j}$ is intermediate input for f_j produced by f_i , where $d_{i,j} \in D$ and $f_i, f_j \in F$.
- iii) T is the set of transformation rules depicting data and control flow dependencies.
- iv) R is the set of physical resources required to realize work-items F .

A workflow W is represented as the set of work-items $f_1, f_2, f_3, \dots, f_k \in F$ having working dataset $d_{i,j}$ where $i, j \leq k$. When f_i does not have a predecessor, it is called as the initiating work-item f_{init} and when f_i does not have any successor, it is called a terminating work-item f_{term} . $d_{init,i}$ is initial data set given as input for starting the workflow and $d_{j,term}$ is the output produced as the result of the workflow enactment. $d_{(1,2,3,\dots,k-1),k}$ shows merging of various data input at f_k . In a special case, $d_{i,j} \mid j = 0$, intermediate result is not consumed by any other work-item $f \in F$. $t_{i,j} \in T$ is the transformation rule applied in order to progress the execution from f_i to f_j . $r_{(l,m,n)} \in R$ is the resource executing f_l work-item with n instances of the resource with id m . In next section, we discuss about the membrane oriented workflow definition and enactment.

3 Membrane Oriented Workflow Definition

The advancements in the field of distributed computing (such as cloud computing, peer-to-peer computing, mobile computing etc.) have enabled several providers to offer services for the realization of workflow in distributed, decentralized and automated manner. These providers are also equipped with the technologies to provide the services elastically. The term “elastic” here stands for the dynamic provisioning of computing resources for the services. Today technology has enabled the workflow executor to specify the computational resource requirements along with the workflow. However formal notation of the same are lagging. Our aim in this paper is to propose a generic framework along with the formal notation using P systems for depicting the decentralized, autonomous and elastic enactment of the workflows. We focus on defining and executing the workflows using P systems. Our definition and execution relies on the membrane vision of every “work-item” or “process step” involved in a workflow. In our approach, the membrane represents the service executing a work-item along with the data they process and possess, control-flow information, resource allotment information in form of evolution rules of membranes. In proposed work, the following types of evolution rules are considered:

- i) Data-flow rules: Rules describing the flow of data for executing the workflow.
- ii) Control-flow rules: Rules describing the flow of execution control for coordination among the various services executing the work-items of the workflow.
- iii) Resource-provision rules: Rules describing the computing resource allotment during the execution of a workflow.

Proposed P system is able to depict the “elasticity”, “parallelism” and “decentralization” in the workflow enactment :

- “Elasticity” of the computing resources is achieved using evolution rules *membrane division* and *membrane merging*. The new membrane ‘forked’ due to membrane division will inherit the dataset and evolution rules of its parent. While the membrane merging will take place only for the membranes depicting the same behavior in terms of dataset and evolution rules. (Discussed in detail in section 3.4)
- “Parallelism” of various work-items can be easily depicted by any of the P system. As membranes in the P system are autonomous and executes in parallel. Hence, P systems are best suitable for formalizing workflow and their execution.
- “Decentralization” in the workflow enactment is achieved by using *object*. We use objects in proposed P system for the communication. These communication carry intermediate data and global data needed for executing a work-item. We assume that there are multiple copies of objects available to the membranes.

3.1 Workflow Definition from Membrane Perspective

Membranes has inspired us to solve the problem of distributed, decentralized and elastic workflow enactment using P system. The membranes are the nature’s way of executing a workflow at cellular level. This has motivated us to envision real world workflow enactment from membrane’s perspective. As the membrane in the cellular bodies functions autonomously, reproduces itself without any central control, similarly a workflow should be realized in decentralized and elastic manner. P system has provided mathematical base for the realization of this vision. From membrane computing’s perspective, we propose a novel and generic definition for the workflow:

Definition 3: A workflow with n work-items is a construct Π :

$$\Pi = (V, C, \mu, (w_1, R_1, R'_1, t_1) \dots (w_i, R_i, R'_i, t_i), i_0)$$

where:

- V is multiset of all objects.
- C is the set of catalyst and catalyst do not occur in t_i .
- μ is the membrane structure consisting of i membranes: $[_n[_{n-1} \dots]_{n-1}]_n$.
- $w_i \in V$ is the multiset of initial contents of region i of μ .
- R_i is the set of data-flow rules.
- R'_i is the set of resource provision and and control-flow.
- t_i is the multiset of final contents of region i of μ .
- i_0 is the label of regions.

In the above definition, we have introduced the notation of a specialized operator, the dependency operator, denoted as $\xrightarrow[\text{Dependency}]{(Details)}$. This operator gives a sense of determinism in the P system. It is utilized while provisioning resources or resolving control dependencies.

With regard to control flow, the operator specifies the dependency among membranes. This control dependency is shown as $\xrightarrow[\text{Control}]{\langle MembraneList \rangle}$, where $\langle MembraneList \rangle$ are the list of membranes that must be executed (or dissolved) first for the execution of the present evolution rule. Proposed dependency operator separate the control rules from the rest of the evolution rules and provides synchronization in workflow execution. For example:

$$[1[S_1 a]_{S_1}[S_2 b]_{S_2}[S_3 c]_{S_3}]_1 \xrightarrow[\text{Control}]{S_2, S_3} [1[S_1 a]_{S_1} \ e \ f \]_1 \delta \quad (5)$$

The above rule states that S_2, S_3 should be executed prior to the S_1 , though all the three membranes S_1, S_2, S_3 are same level. It is a known fact that all the membranes at same level (part of the same parent membrane 1) get executed in parallel and in a non-deterministic manner. However, while accomplishing a workflow few membranes are required to be restricted from execution (in case of synchronization). In such scenarios, the proposed symbol gives a certain level of determinism and restricts few of the membranes from execution.

In case of resource provision rules, dependency operator states the dependency of resources for enacting a work-item. Resource dependency is of the form $\xrightarrow[\text{Resource}]{\langle Dependencyparameter \rangle}$. Proposed operator enables the specification of the resources requirements for a work-item, along with the workflow definition. Dependency operator enables the elastic workflow execution under predefined resource requirements (as in the case of most of the modern pay-as-you-go technologies e.g. cloud computing) (Resource dependency is discussed in detail in Section 3.3). To the best of our knowledge, we are the first to classify the data flow, control flow and resource flow perspectives in a low level formal workflow definition.

Few of the features (of the proposed formal notation) apart from the discussed *elasticity, parallelism and decentralization* are:

- Reliable Data Exchange: This is usually achieved by stable communication. In proposed definition, this stable communication and data exchange is shown by inter-membrane communication via object passing.
- Heterogeneous Environment: Our approach of workflow states the workflow definition and workflow enactment at the abstract layer. The execution level details for the enactment, such as executing resources are open ended. These execution resources could be web services, computing devices, mobile devices or human.
- Fault Tolerance: Our approach is fault tolerant for workflow enactment. A failed membrane may produce erroneous result that could be rejected by the next membrane in workflow. Hence, resulting in the redundant execution of the membrane. However, a more sophisticated function can be introduced

$Rollback(\Pi, C_v)$. Rollback function is able to trace back to last valid configuration C_v of the P system Π .

Before proceeding any further, first we must justify the proposed definition for a workflow. In proposed definition, V is the multiset of all the objects including initial input to the workflow, intermediate processing dataset etc. C also termed as the catalyst is the set of all the dataset and conditions that remain in the membrane after the execution. The catalyst does not add to the final contents or result, they help in faster execution of the work-item. μ is the set of all the membranes or work-items with the resources in the workflow. w_n is the initial state of the membranes with which they start executing. t_n is the final content of the membrane once its execution is completed. Hence, for defining the workflow enactment in terms of P system, these are the minimal required elements. In the move to execute membranes or work-items on the resources, evolution rules are required. The definition shows two types of evolution rules: R_n - Data Rules and R'_n - Resource Provision and control flow Rules. Expressive power of P systems enables them to specify both data-flow and control-flow rules in a single evolution rule.

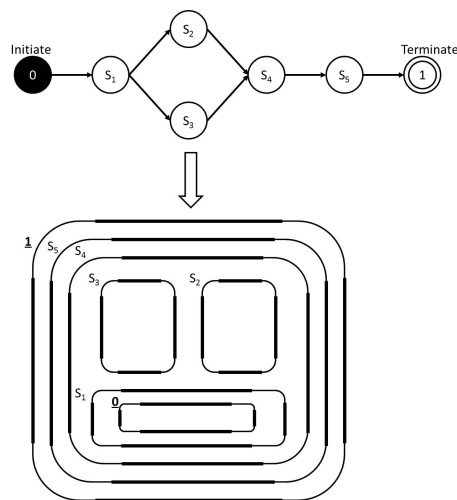


Fig. 3. A Simple Workflow and its Membrane Representation

Figure 3 represents a simple workflow and its corresponding membrane representation in a graphical manner. The same workflow can also be represented as:

$$[1[s_5[s_4[s_3[s_2[s_1[0]0]s_1]s_4]s_5]1 \tag{6}$$

3.2 Solving Workflow Pattern using Membrane Computing

There are various workflow patterns [4] which we have solved using the P systems, mainly utilizing membrane evolution rules. Few of the basic patterns are:

Sequence: Any work-item in the workflow is executed only after the completion of its predecessor work-item. Evolution rules depicting the sequence pattern involve dissolution operation after each successful enactment of a work-item/membrane (as shown in rule (7), (8), (9)).

$$[1[s_3[s_2[s_1 a]s_1]s_2]s_3]_1 \rightarrow [1[s_3[s_2 b]s_2]s_3]_1 \delta \quad (7)$$

$$[1[s_3[s_2 b]s_2]s_3]_1 \rightarrow [1[s_3 c]s_3]_1 \delta \quad (8)$$

$$[1[s_3 c]s_3]_1 \rightarrow [1 d]_1 \delta \quad (9)$$

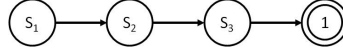


Fig. 4. Sequential Execution of Work-items

Parallel Split: A parallel split is the point in a workflow where, a single work-item feeds the data/control flow to multiple work-items. These multiple work-items execute in parallel. Rule (10), (11) shows the parallel split:

$$[1[s_1 a]s_1[s_2]s_2[s_3]s_3]_1 \rightarrow [1[s_2 b]s_2[s_3 b]s_3]_1 \delta \quad (10)$$

$$[1[s_2 b]s_2[s_3 b]s_3]_1 \rightarrow [1 c d]_1 \quad (11)$$

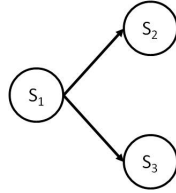


Fig. 5. Parallel Split of Work-items

Synchronization: Synchronization in the workflow is the point when multiple parallel work-items converge onto a single work-item. A condition involved here is that all the parallel multiple work-items (that are converging) should have been executed only once by following a certain order. Rules (12), (13), (14), (15) shows the synchronization pattern:

$$[1[s_1 a]s_1[s_2 b]s_2[s_3 c]s_3[s_4]s_4]_1 \xrightarrow[Control]{S_1} [1[s_2 b]s_2[s_3 c]s_3[s_4 d]s_4]_1 \delta \quad (12)$$

$$[1[s_2b]s_2[s_3c]s_3[s_4d]s_4]1 \xrightarrow[\text{Control}]{S_2} [1[s_3c]s_3[s_4de]s_4]1\delta \quad (13)$$

$$[1[s_3c]s_3[s_4de]s_4]1 \rightarrow [1[s_4def]s_4]1\delta \quad (14)$$

$$[1[s_4def]s_4]1 \rightarrow [1g]1\delta \quad (15)$$

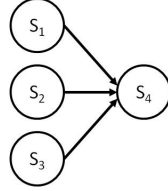


Fig. 6. Synchronization of Work-items

Exclusive Choice: This workflow pattern is to specify the decision condition, where one of the several incoming work-items is chosen based on some condition. Rule (16) states that service S_4 accepts d from the predecessor services S_1 , S_2 , S_3 , where d is satisfying certain condition, which is not satisfied by e and f . Here $k_1 \geq d \geq k_2$ where k_1 and k_2 are integers.

$$\begin{aligned} & [1[s_1a]s_1[s_2b]s_2[s_3c]s_3[s_4]s_4]1 \rightarrow [1[s_4d]s_4[ef]]1\delta \\ & \left| \begin{array}{l} d := \{k_1 \geq x \geq k_2 \mid x \in \{a, b, c\}\} \text{ and } e \neq d \text{ and } f \neq d \end{array} \right. \quad (16) \end{aligned}$$

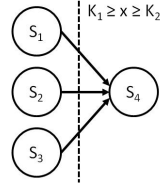


Fig. 7. Exclusive Choice from Work-items

Simple Merge: Merge pattern specifies that two or more work-items are converging onto a single work-item without any synchronization.

$$[1[s_1a]s_1[s_2b]s_2[s_3]s_3]1 \rightarrow [1[s_2b]s_2[s_3d]s_3]1\delta \quad (17)$$

$$[1[s_2b]s_2[s_3d]s_3]1 \rightarrow [1[s_{3*}f]s_{3*}[e]]1\delta \quad (18)$$

$$[1[s_{3*}f]s_{3*}[e]]1 \rightarrow [1ge]1\delta \quad (19)$$

Rules (17), (18) and (19) shows the merging of work-items S_1 and S_2 in work-item S_3 . Rule (18) has the dissolution and division of S_3 , this states the non-parallel execution of work-items S_2 and S_1 . (Symbol \star is used to show the membrane left after division and dissolution).

3.3 Resource-Oriented Workflow Enactment

The present technology has enabled a client to specify the process steps as well as resource dependencies at the same time. As pay-as-you-go models, such as cloud computing, are becoming more popular, the client has the ability to specify the type of resources, the exact specifications, the requirement and several others needed to successfully execute a workflow. These resource requirements are depicted by resource provision rules R_i , (as shown in the definition 3). We propose to use the symbol $\xrightarrow[Resource]{(S,r,o)}$ for specifying need of the requester:

Resource Dependency: Any work-item requires a physical resource such as service, physical devices etc. for execution. In our case, a membrane may depend on other membranes for the execution. For example, requester might need to execute the work-item on a specific membrane/resource. The symbol S shows the dependency on S for executing an evolution rule.

Resources Requirement: The symbol r states two types of requirements: Qualitative and Quantitative. Qualitative requirements are the non-functional requirements for the resource S , such as reliability requirement and other QoS (Quality of Service) parameters. While the quantitative requirements are the functional requirements (r) such as memory requirements, processor requirement, number of parallel threads of execution.

Resource Ownership: The symbol o shows the ownership requirement of the resource. In case of shared resources, the requirement depicts the dedicated need for a resource. This specifies the requirement for the ownership change of the shared resource.

Suppose a sequential workflow comprises of three work-items S_1, S_2, S_3 . Workflow will only be realized when the second work-item S_2 satisfy additional qualitative requirements (x_1, x_2, x_3) (e.g. $x_1 = \text{OS: Linux}$, $x_2 = \text{RAM: 2GB}$, $x_3 = \text{MEM: 50GB}$). Moreover, for the execution purpose ownership of the resource should be given to first work-item S_1 . This can be shown in rule (20):

$$\begin{aligned} & [1[S_3[S_2[S_1 a]S_1]S_2]S_3]_1 \xrightarrow[Resource]{(S,r,o)} [1[S_3[S_2 \ b]S_2]S_3]_1 \delta \\ & \left| \begin{array}{l} S \in \{S_2\}, r \in \{x_1, x_2, x_3\}, o \in \{S_1\} \end{array} \right. \end{aligned} \quad (20)$$

3.4 Elastic Workflow Enactment

Elastic workflow enactment is the unique feature of our model. There has been several work on workflow execution in the literature, however, to the best of our knowledge we are the first to investigate this through membrane computing paradigm. Elastic execution enables the workflow for dynamic procurement of computing resources. This enables the pay-as-you-go models of current technologies on the workflow enactment as well as the load balancing by dynamic provisioning of computing resources. We propose to use membrane division rule for depicting elasticity in the formal notation of workflow.

Consider a membrane is executing a resource intensive task. In the middle of the execution, the load-indicator sensed that new computing resource must be provisioned in order to balance the load. Such a scenario can be shown by evolution rule comprising of membrane division. Furthermore, the advancements in technology has enabled these computing resources to procure extra resources as per requirement. In the proposed work, the membrane is divided into multiple membrane depicting the behavior of dynamic provisioning of computing resource. All the child membranes, resulting from membrane division, have the same set of objects and evolution rules. Hence, children membrane mimics the behavior of the parent membrane.

Suppose a workflow comprises of three work-items S_1, S_2, S_3 . Work-item S_2 is resource intensive and require two extra replica of it. This can be shown using the resource dependency operator with quantitative resource requirements.

$$[1[S_3[S_2[S_1 a]S_1]S_2]S_3]_1 \xrightarrow[\text{Resource}]{(S,r,o)} [1[S_3[S_2 \ b \]S_2[S_2' \ b \]S_2'[S_2'' \ b \]S_2'']S_3]_1 \delta \quad (21)$$

$$\left| \begin{array}{l} S \in \{S_2\}, r \in \{3\}, o \in \{S_1\} \end{array} \right.$$

4 Related Work

There are numerous techniques available in the literature on workflow management and enactment. However, decentralized workflow enactment and dynamic resource provisioning in workflow is rarely targeted. In related work section, we present a brief discussion on these techniques and models.

Fernandez et al [6] propose executing a workflow using the chemical paradigm. Similar to our work, the authors used a decentralized environment, however, they used a centralized shared memory (hence, the authors suffered from scalability issues). Moreover, they kept the services fixed to execute the work-items, with no provision of dynamic adaptations. Further, the issues of elasticity is not addressed in this work. Another work by Caeiro et al [1] discuss about the dynamic workflow enactment using chemical analogy. The authors presents a generalized notion of workflow representation using HOCL (Higher Order Chemical Language). Another work by Németh et al [10] present modern workflow execution on large scale computing devices and propose an enactment model

using gamma calculus. In [14] Weske discuss about the different aspects of flexible workflow management and presents workflow schema using object-oriented approach. There are several literature available on workflow specifications by Leymann et. al. [8], Hollingsworth [7], Aalst et.al. [13], Weske [14].

However, discussed work lacks an important feature of today's modern workflow, that is elasticity. Elastic execution of the workflow and its formal notation is not yet discussed and investigated. To the best of our knowledge we are the first to look into elastic execution of the workflow from membrane computing paradigm perspective.

5 Conclusion and Future Work

In this paper, we introduced the membrane computing paradigm for workflow enactment. We presented a generic workflow model for various types of workflows ranging from business, scientific, mobile, cloud etc. The proposed model is specially designed for distributed, decentralized and elastic execution of workflow. Our model is closely related to the actual realization of workflow on the real computing resources. Furthermore, we proposed dependency specific operators for the workflow enactment notation: Resource Dependency Operator and Control Dependency Operator. Moreover, we discussed these operators in detail. Along with this, we discussed a prospective fault tolerant function "*Rollback*".

The future work include extension of the proposed model for case specific scenario such as volatile mobile environment. Also, to study the behavior of dynamic and faulty resources on the workflow. Finally, include various real world faults and errors in the move to make the notation fault tolerant and robust.

Acknowledgment

The authors would like to thank fellow colleagues, to help with the technical issues and fruitful discussion on the various aspects of Membrane Computing. We would also like to thank G. Păun and all the people in research community for sharing their research works.

References

1. Caeiro, M., Németh, Z., Priol, T.: A chemical model for dynamic workflow coordination. In: Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on. pp. 215–222. IEEE (2011)
2. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J.: Applications of membrane computing, vol. 17. Springer (2006)
3. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., et al.: Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing* 1(1), 25–39 (2003)
4. van Der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. *Distributed and parallel databases* 14(1), 5–51 (2003)

5. Dornemann, T., Juhnke, E., Freisleben, B.: On-demand resource provisioning for BPEL workflows using amazon's elastic compute cloud. In: Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on. pp. 140–147. IEEE (2009)
6. Fernandez, H., Tedeschi, C., Priol, T.: A chemistry-inspired workflow management system for a decentralized workflow execution. In: IEEE Transactions on Services Computing. vol. PP, pp. 1–1 (2013)
7. Hollingsworth, D.: The workflow reference model. Workflow Management Coalition Document Number TC00-1003 (1995)
8. Leymann, F., Roller, D.: Production workflow: concepts and techniques. Prentice Hall (2000)
9. Lin, C., Lu, S.: SCPOR: An elastic workflow scheduling algorithm for services computing. In: Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on. pp. 1–8. IEEE (2011)
10. Németh, Z., Pérez, C., Priol, T.: Distributed workflow coordination: molecules and reactions. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. pp. 8. IEEE (2006)
11. Păun, G.: Computing with membranes. *Journal of Computer and System Sciences* 61(1), 108–143 (2000)
12. Păun, G., Pérez-Jiménez, M.J.: Membrane computing: Brief introduction, recent results and applications. *BioSystems* 85(1), 11–22 (2006)
13. Van Der Aalst, W.M., Ter Hofstede, A.H.: Yawl: yet another workflow language. *Information systems* 30(4), 245–275 (2005)
14. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on. pp. 10. IEEE (2001)

Fault Diagnosis Models for Electric Locomotive Systems Based on Fuzzy Reasoning Spiking Neural P Systems

Tao Wang¹, Gexiang Zhang¹ and Mario J. Pérez-Jiménez² *

¹School of Electrical Engineering, Southwest Jiaotong University,
Chengdu, 610031, P.R. China

email: wangtaocdu@gmail.com, zhgxdylan@126.com

²Research Group on Natural Computing

Department of Computer Science and Artificial Intelligence

University of Sevilla, Sevilla, 41012, Spain

email: marper@us.es

Abstract. This paper discusses the application of fuzzy reasoning spiking neural P systems with real numbers (rFRSN P systems) to fault diagnosis of electric locomotive systems. Relationships among breakdown signals and faulty sections in subsystems of electric locomotive systems are described in the form of fuzzy production rules firstly and then fault diagnosis models based on rFRSN P systems for these subsystems are built according to these rules. Fuzzy production rules for diagnosing electric locomotive systems are abstracted via the analysis of fault diagnosis models for subsystems and the causality among faulty sections, faulty subsystems and electric locomotive systems. Finally, a diagnosis model based on rFRSN P systems for electric locomotive systems is proposed.

Keywords: fuzzy reasoning spiking neural P system, fault diagnosis, electric locomotive system, real number, SS4 electric locomotive systems

1 Introduction

Membrane computing, introduced by Gh. Păun in [1], is a class of distributed parallel computing models inspired by the structure and functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures. Spiking neural P systems (SN P systems), introduced in [2] in the framework of membrane computing, is a new class of computing devices which are inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons. Since then, SN P systems have become a hot topic in membrane computing [3]-[21], among which there are several investigations focus on the use of SN P systems and their variants to solve engineering problems in power systems [18]-[21].

* Corresponding author.

In [18], fuzzy reasoning spiking neural P systems with real numbers (rFRSN P systems) were presented to fulfill diagnosis knowledge representation and reasoning. The merits of rFRSN P systems lie in visually describing fuzzy production rules in a fuzzy diagnosis knowledge base and effectively modeling the relationships among breakdown signals and faulty sections as well as representing and handling fuzzy knowledge/information. To extend application areas of SN P systems and rFRSN P systems, this paper discusses the application of rFRSN P systems to fault diagnosis of Shaoshan4 (SS4) electric locomotive systems. In this paper, electric locomotive systems always indicate SS4 electric locomotive systems.

Electric locomotive systems are composed of several subsystems with different functions; meanwhile, these subsystems consist of lots of sections. Thus, a locomotive system can be viewed as a hierarchical tree structure of sections and subsystems [22,23]. To build fault diagnosis models based on rFRSN P systems for different subsystems, relationships among breakdown signals and faulty sections in subsystems are abstracted and described in the form of fuzzy production rules firstly. Then, fault diagnosis models for these subsystems are built according to these rules. According to analyzing the causality among faulty sections, faulty subsystems and SS4 electric locomotive systems, fuzzy production rules for diagnosing electric locomotive systems are abstracted and then a diagnosis model based on rFRSN P systems for SS4 electric locomotive systems is proposed. Moreover, it is worth pointing out that rule neurons in rFRSN P systems used in this paper are extended according to the problem to be solved. In other words, rFRSN P systems used in this paper contained three types of rule neurons, i.e., *GENERAL*, *AND* and *OR*, while the ones in [18] only contain two types: *AND* and *OR*.

The remainder of this paper is organized as follows. Section 2 gives preliminaries of this work. The fault diagnosis models for key subsystems and electric locomotive systems are presented in Section 3. Conclusions are finally drawn in Section 4.

2 Preliminaries

In this section, we briefly review the basic concepts of fuzzy reasoning spiking neural P systems with real numbers (rFRSN P systems) [18]. Here, only the necessary prerequisites are introduced.

Definition 1: An rFRSN P system of degree $m \geq 1$ is a construct $\Pi = (O, \sigma_1, \dots, \sigma_m, syn, in, out)$, where:

- (1) $O = \{a\}$ is a singleton alphabet (a is called spike);
- (2) $\sigma_1, \dots, \sigma_m$ are neurons, of the form $\sigma_i = (\theta_i, c_i, r_i)$, $1 \leq i \leq m$, where:
 - (a) θ_i is a real number in $[0, 1]$ representing the potential value of spikes (i.e. value of electrical impulses) contained in neuron σ_i ;
 - (b) c_i is a real number in $[0, 1]$ representing the fuzzy truth value corresponding to neuron σ_i ;

- (c) r_i represents a firing (spiking) rule contained in neuron σ_i with the form $E/a^\theta \rightarrow a^\beta$, where E is the firing condition and its form will be specified below, θ and β are real numbers in $[0, 1]$;
- (3) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for all $(i, j) \in syn$, $1 \leq i, j \leq m$, is a directed graph of synapses between the linked neurons;
- (4) $in, out \subseteq \{1, 2, \dots, m\}$ indicate the input neuron set and the output neuron set of Π , respectively.

In rFRSN P systems, each neuron associates with either a fuzzy proposition or a fuzzy production rule, and $c_i \in [0, 1]$ is used to express truth value of this fuzzy proposition or confidence factor (CF) of this fuzzy production rule. Each neuron contains only one firing (spiking) rule, of the form $E/a^\theta \rightarrow a^\beta$, where $E = a^n$ is called the firing condition and n is the number of input synapses from other neurons to the neuron. The firing condition $E = a^n$ means that the spiking rule, $E/a^\theta \rightarrow a^\beta$, contained in neuron σ_i , can be applied if and only if neuron σ_i contains at least n spikes, otherwise, the rule cannot be enabled until n spikes are received. For neuron σ_i , if its firing rule is applied, then its pulse value θ_i is consumed (removed) and a new spike with value β is produced in σ_i . Once the spike with value β is emitted from neuron σ_i , each neuron σ_j with $(i, j) \in syn$ immediately receives this spike.

Moreover, the definitions of neurons and pulse values are extended. Specifically, in rFRSN P systems, the neurons are extended to four types, i.e., proposition neurons and three kinds of rule neurons: *GENERAL*, *AND* and *OR*, and the pulse value contained in each neuron is no longer the number of spikes represented by a real value, but a real number in $[0, 1]$ representing the potential value of spikes contained in neuron σ_i . For neuron σ_i , if $\theta_i > 0$, then the neuron contains a spike with pulse value θ_i ; otherwise, their neuron contains no spike and its pulse is 0. For different types of neurons, their definitions and the operations for pulse values are different. Since *GENERAL* rule neurons is a new type in this paper, we described its definition as follows and more details about other three types of neurons can be found in the work [18].

Definition 2: A *GENERAL* rule neuron is associated with a fuzzy production rule which has only one proposition in the antecedent part of the rule. Such a neuron is represented by a rectangle, as shown is Fig. 1.

A *GENERAL* rule neuron has only one presynaptic proposition neuron and one or more postsynaptic proposition neurons. If a *GENERAL* rule neuron receives a spike from its presynaptic proposition neuron and its firing condition is satisfied, then the neuron fires and produces a new spike with the potential value $\beta = \theta * c$, where β , θ and c are real numbers in $[0, 1]$.

3 Fault Diagnosis Models for Electric Locomotive Systems Based on rFRSN P Systems

In this section, fault diagnosis models based on rFRSN P systems for SS4 electric locomotive systems and their main subsystems, i.e., main circuit systems,

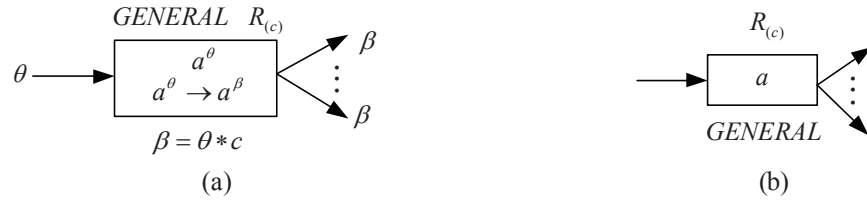


Fig. 1. A *GENERAL* rule neuron (a) and its simplified form (b).

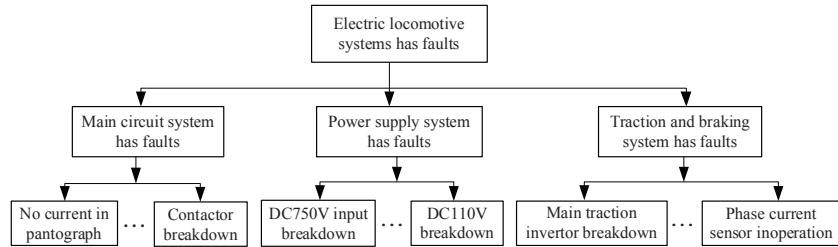


Fig. 2. A hierarchial tree structure of components in an SS4 electric locomotive system

power supply systems, traction and breaking systems, are proposed. Since an SS4 electric locomotive system can be viewed as a hierarchial tree structure of subsystems and sections shown in Fig. 2, we can build the diagnosis models from leaves to the top (that is, the root) of the tree. Thus, we firstly build the models for subsystems and then analyze these models and relationships among an electric locomotive system, its subsystems and faulty sections in these subsystems. Finally, a fault diagnosis model for SS4 electric locomotive systems is proposed.

3.1 A fault diagnosis model for the main circuit systems

Fuzzy production rules (*Rules 1 to 16*), describing the relationships between breakdown signals detected and candidate faulty sections, for main circuit systems of electric locomotives are described as follows, where CF is an empirical value representing the certainty (confidence) factor of a rule, P_1, \dots, P_{18} are propositions whose meanings are shown in Table 1. According to these fuzzy production rules, a fault diagnosis model based on rFRSN P systems for main circuit systems Π_1 is built, as shown in Fig. 3. $\Pi_1 = (O, \sigma_1, \dots, \sigma_{34}, syn, in, out)$ where

- (1) $O = \{a\}$ is the singleton alphabet (a is called spike);
- (2) $\sigma_1, \dots, \sigma_{18}$ are proposition neurons associated with propositions P_1, \dots, P_{18} , respectively;
- (3) $\sigma_{19}, \dots, \sigma_{28}, \sigma_{30}, \dots, \sigma_{34}$ are *GENERAL* rule neurons associated with fuzzy production rules $R_1, \dots, R_{10}, R_{12}, \dots, R_{16}$, respectively; σ_{29} is an *OR* rule neuron associated with fuzzy production rule R_{11} ;

Table 1. Meaning of each proposition in fuzzy production rules for main circuit systems

P_1	pantograph bounce	P_{10}	nonlinear resistor sparkwear
P_2	mechanical part damaged	P_{11}	coil/main contact sparkwear
P_3	scratching of pantograph	P_{12}	contact in poor contact
P_4	insulating oil damp	P_{13}	no current in pantograph
P_5	cooling system breakdown	P_{14}	traction transformer breakdown
P_6	transformer internal breakdown	P_{15}	pulling motor inoperation
P_7	electrical strength reduction	P_{16}	circuit breaker inoperation
P_8	electromotor overload	P_{17}	contactor breakdown
P_9	isolating switch sparkwear	P_{18}	main circuit breakdown

- (4) $syn = \{(1, 19), (2, 20), (3, 21), (4, 22), (5, 23), (6, 24), (7, 25), (8, 26), (9, 27), (10, 28), (11, 29), (12, 29), (13, 30), (14, 31), (15, 32), (16, 33), (17, 34), (19, 13), (20, 13), (21, 13), (22, 14), (23, 14), (24, 14), (25, 15), (26, 15), (27, 16), (28, 16), (29, 17), (30, 18), (30, 15), (31, 18), (31, 15), (32, 18), (33, 15), (34, 18)\}$;
- (5) $in = \{\sigma_1, \dots, \sigma_{12}\}$, $out = \{\sigma_{18}\}$.

Rule 1: IF P_1 THEN P_{13} (CF=0.95)

Rule 2: IF P_2 THEN P_{13} (CF=0.85)

Rule 3: IF P_3 THEN P_{13} (CF=0.9)

Rule 4: IF P_4 THEN P_{14} (CF=0.8)

Rule 5: IF P_5 THEN P_{14} (CF=0.85)

Rule 6: IF P_6 THEN P_{14} (CF=0.8)

Rule 7: IF P_7 THEN P_{15} (CF=0.95)

Rule 8: IF P_8 THEN P_{15} (CF=0.8)

Rule 9: IF P_9 THEN P_{16} (CF=0.95)

Rule 10: IF P_{10} THEN P_{16} (CF=0.9)

Rule 11: IF P_{11} OR P_{12} THEN P_{17} (CF=0.9)

Rule 12: IF P_{13} THEN P_{15} AND P_{18} (CF=1.0)

Rule 13: IF P_{14} THEN P_{15} AND P_{18} (CF=0.9)

Rule 14: IF P_{15} THEN P_{18} (CF=0.9)

Rule 15: IF P_{16} THEN P_{15} (CF=0.85)

Rule 16: IF P_{17} THEN P_{18} (CF=0.85)

3.2 A fault diagnosis model for the power supply systems

Fuzzy production rules (*Rules 1 to 13*), describing the relationships between breakdown signals detected and candidate faulty sections, for power supply systems of electric locomotives are described as follows, where CF is an empirical value representing the certainty (confidence) factor of a rule, P_1, \dots, P_{17} are propositions whose meanings are shown in Table 2. According to these fuzzy

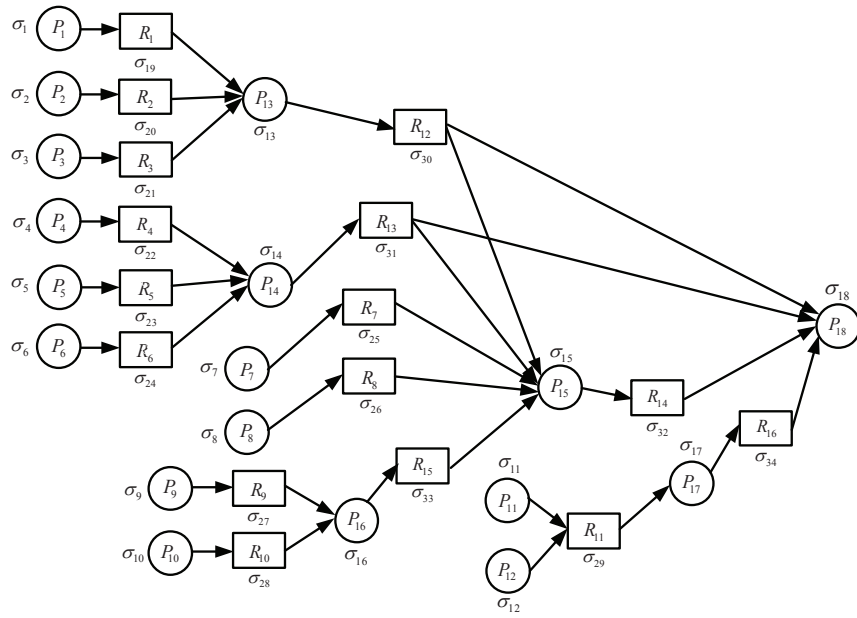


Fig. 3. A fault diagnosis model based on rFRSN P systems for the main circuit systems

production rules, a fault diagnosis model based on rFRSN P systems for power supply systems Π_2 is built, as shown in Fig. 4. $\Pi_2 = (O, \sigma_1, \dots, \sigma_{30}, syn, in, out)$ where

- (1) $O = \{a\}$ is the singleton alphabet (a is called spike);
- (2) $\sigma_1, \dots, \sigma_{17}$ are proposition neurons associated with propositions P_1, \dots, P_{17} , respectively;
- (3) $\sigma_{18}, \dots, \sigma_{21}, \sigma_{23}, \sigma_{24}, \sigma_{26}, \dots, \sigma_{30}$ are *GENERAL* rule neurons associated with fuzzy production rules $R_1, \dots, R_4, R_6, R_7, R_9, \dots, R_{13}$, respectively; σ_{22} is an *OR* rule neuron associated with fuzzy production rule R_5 ; σ_{25} is an *AND* rule neuron associated with fuzzy production rule R_8 ;
- (4) $syn = \{(1, 18), (2, 19), (3, 20), (4, 21), (5, 22), (6, 23), (7, 24), (8, 25), (9, 26), (10, 27), (11, 22), (12, 28), (13, 25), (14, 29), (16, 30), (18, 17), (19, 10), (20, 10), (21, 11), (22, 12), (23, 14), (23, 13), (24, 13), (25, 15), (26, 16), (27, 17), (27, 11), (27, 13), (27, 14), (28, 14), (28, 17), (29, 16), (30, 17)\}$;
- (5) $in = \{\sigma_1, \dots, \sigma_9\}$, $out = \{\sigma_{15}, \sigma_{17}\}$.

- Rule 1:* IF P_1 THEN P_{17} (CF=0.9)
- Rule 2:* IF P_2 THEN P_{10} (CF=0.8)
- Rule 3:* IF P_3 THEN P_{10} (CF=0.85)
- Rule 4:* IF P_4 THEN P_{11} (CF=0.9)
- Rule 5:* IF P_5 OR P_{11} THEN P_{12} (CF=0.85)
- Rule 6:* IF P_6 THEN P_{13} AND P_{14} (CF=0.8)

Table 2. Meaning of each proposition in fuzzy production rules for power supply systems

P_1	main traction inverter breakdown	P_9	fan breakdown
P_2	current collector breakdown	P_{10}	DC750V input breakdown
P_3	locomotive current collector breakdown	P_{11}	110V DC/DC chopper inoperation
P_4	110V DC/DC chopper breakdown	P_{12}	DC110V breakdown
P_5	110V accumulator breakdown	P_{13}	280V DC/DC chopper inoperation
P_6	auxiliary inverter breakdown	P_{14}	auxiliary inverter inoperation
P_7	280V DC/DC chopper breakdown	P_{15}	DC280V breakdown
P_8	280V accumulator breakdown	P_{16}	fan inoperation
P_{17}	main traction inverter inoperation		

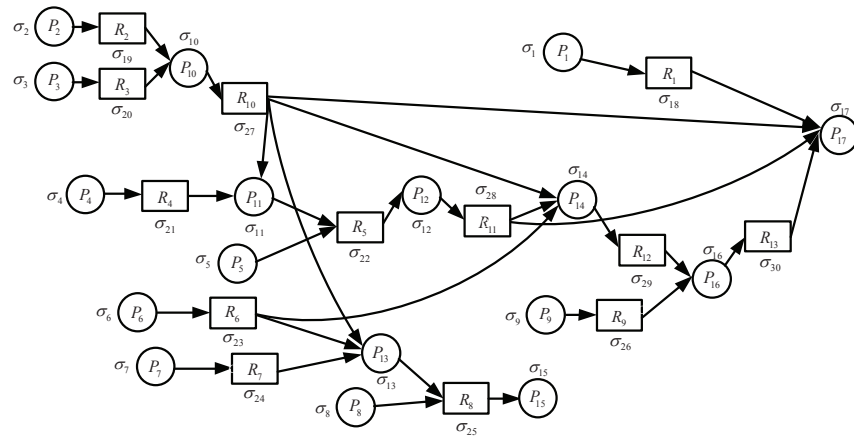


Fig. 4. A fault diagnosis model based on rFRSN P systems for the power supply systems

Rule 7: IF P_7 THEN P_{13} (CF=0.95)

Rule 8: IF P_8 AND P_{13} THEN P_{15} (CF=0.85)

Rule 9: IF P_9 THEN P_{16} (CF=0.9)

Rule 10: IF P_{10} THEN P_{11} AND P_{13} AND P_{14} AND P_{17} (CF=0.85)

Rule 11: IF P_{12} THEN P_{14} AND P_{17} (CF=0.8)

Rule 12: IF P_{14} THEN P_{16} (CF=0.95)

Rule 13: IF P_{16} THEN P_{17} (CF=0.9)

3.3 A fault diagnosis model for the traction and braking systems

Fuzzy production rules (*Rules 1 to 10*), describing the relationships between breakdown signals detected and candidate faulty sections, for traction and braking systems of electric locomotives are described as follows, where CF is an empirical value representing the certainty (confidence) factor of a rule, P_1, \dots, P_{16} are propositions whose meanings are shown in Table 3. According to these fuzzy production rules, a fault diagnosis model based on rFRSN P systems for traction and braking systems Π_3 is built, as shown in Fig. 5. $\Pi_3 = (O, \sigma_1, \dots, \sigma_{26}, syn, in, out)$ where

- (1) $O = \{a\}$ is the singleton alphabet (a is called spike);
- (2) $\sigma_1, \dots, \sigma_{16}$ are proposition neurons associated with propositions P_1, \dots, P_{16} , respectively;
- (3) $\sigma_{17}, \sigma_{18}, \sigma_{23}, \dots, \sigma_{26}$ are *GENERAL* rule neurons associated with fuzzy production rules $R_1, R_2, R_7, \dots, R_{10}$, respectively; $\sigma_{20}, \dots, \sigma_{22}$ are *OR* rule neurons associated with fuzzy production rules R_4, \dots, R_6 , respectively;
- (4) $syn = \{(1, 17), (2, 18), (3, 19), (4, 19), (5, 19), (6, 20), (7, 20), (8, 21), (9, 22), (10, 23), (11, 22), (12, 24), (13, 25), (14, 26), (15, 21), (17, 16), (18, 16), (19, 13), (20, 14), (21, 16), (22, 16), (23, 16), (24, 16), (25, 15), (26, 16)\}$;
- (5) $in = \{\sigma_1, \dots, \sigma_{12}\}$, $out = \{\sigma_{16}\}$.

Rule 1: IF P_1 THEN P_{16} (CF=0.85)

Rule 2: IF P_2 THEN P_{16} (CF=0.8)

Rule 3: IF P_3 OR P_4 OR P_5 THEN P_{13} (CF=0.9)

Rule 4: IF P_6 OR P_7 THEN P_{14} (CF=0.95)

Rule 5: IF P_8 OR P_{15} THEN P_{16} (CF=0.9)

Rule 6: IF P_9 OR P_{11} THEN P_{16} (CF=0.8)

Rule 7: IF P_{10} THEN P_{16} (CF=0.7)

Rule 8: IF P_{12} THEN P_{16} (CF=0.85)

Rule 9: IF P_{13} THEN P_{15} (CF=0.95)

Rule 10: IF P_{14} THEN P_{16} (CF=0.75)

3.4 A fault diagnosis model for the electric locomotive systems

From Fig. 3 to Fig. 5, we know that if there is no current in pantograph or traction transformers breakdown or pulling motors inoperation or contactors breakdown, then the main circuit system of electric locomotive systems has faults; if DC750V input breakdown or fan inoperation or main traction inverter breakdown or DC110V breakdown, then the power supply system of electric locomotive systems has faults; if DC750V input breakdown or fan inoperation or main traction inverter breakdown or DC110V breakdown, then the power supply system of electric locomotive systems has faults; if the main traction inverter breakdown or DC110V breakdown or control source converter plate breakdown or main protective relay breakdown or traction power controller breakdown or A/D breakdown or 25/5V breakdown or linear electromotor inoperation or phase

Table 3. Meaning of each proposition in fuzzy production rules for traction and braking systems

P_1	DC110V breakdown	P_9	main protective relay breakdown
P_2	control source converter plate breakdown	P_{10}	traction power controller breakdown
P_3	U-phase current sensor breakdown	P_{11}	A/D breakdown
P_4	W-phase current sensor breakdown	P_{12}	25/5V breakdown
P_5	V-phase current sensor breakdown	P_{13}	more than one among P_3, P_4 and P_5 happen
P_6	first linear electromotor group breakdown	P_{14}	linear electromotor inoperation
P_7	second linear electromotor group breakdown	P_{15}	phase current sensor inoperation
P_8	main traction inverter breakdown	P_{16}	traction and braking system inoperation

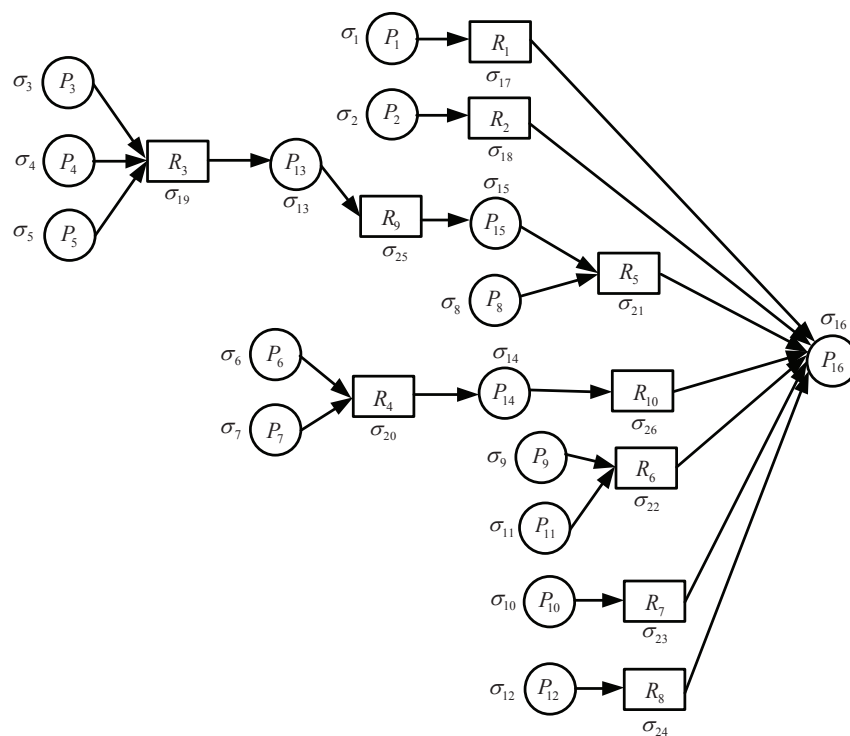


Fig. 5. A fault diagnosis model based on rFRSN P systems for the traction and braking systems

current sensor inoperation, then the traction and braking system of electric locomotive systems has faults.

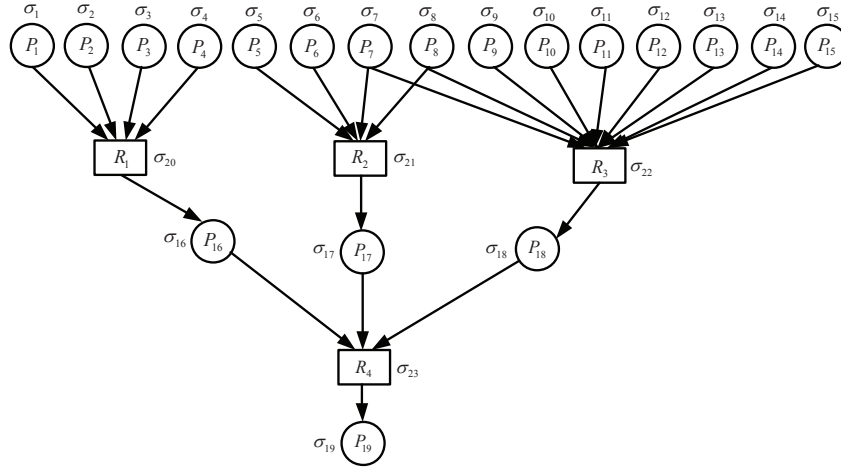


Fig. 6. A fault diagnosis model based on rFRSN P systems for the electric locomotive systems

For an electric locomotive system, if one or more than one of its subsystems (main circuit systems, power supply systems, and traction and braking systems) have faults, then this electric locomotive system has faults. According to the analysis, fuzzy production rules (*Rules 1 to 4*) for electric locomotive systems are described as follows, where CF is an empirical value representing the certainty (confidence) factor of a rule, P_1, \dots, P_{19} are propositions whose meanings are shown in Table 4. According to these fuzzy production rules, a fault diagnosis model based on rFRSN P systems for electric locomotive systems Π_4 is built, as shown in Fig. 6. $\Pi_4 = (O, \sigma_1, \dots, \sigma_{23}, syn, in, out)$ where

- (1) $O = \{a\}$ is the singleton alphabet (a is called spike);
- (2) $\sigma_1, \dots, \sigma_{19}$ are proposition neurons associated with propositions P_1, \dots, P_{19} , respectively;
- (3) $\sigma_{20}, \dots, \sigma_{23}$ are OR rule neurons associated with fuzzy production rules R_1, \dots, R_4 , respectively;
- (4) $syn = \{(1, 20), (2, 20), (3, 20), (4, 20), (5, 21), (6, 21), (7, 21), (7, 22), (8, 21), (8, 22), (9, 22), (10, 22), (11, 22), (12, 22), (13, 22), (14, 22), (15, 22), (16, 23), (17, 23), (18, 23), (20, 16), (21, 17), (22, 18), (23, 19)\}$;
- (5) $in = \{\sigma_1, \dots, \sigma_{15}\}$, $out = \{\sigma_{19}\}$.

Rule 1: IF P_1 OR P_2 OR P_3 OR P_4 THEN P_{16} (CF=0.95)

Rule 2: IF P_5 OR P_6 OR P_7 OR P_8 THEN P_{17} (CF=0.95)

Rule 3: IF P_7 OR P_8 OR P_9 OR P_{10} OR P_{11} OR P_{12} OR P_{13} OR P_{14} OR P_{15} THEN P_{18} (CF=0.95)

Rule 4: IF P_{16} OR P_{17} OR P_{18} THEN P_{19} (CF=0.98)

Table 4. Meaning of each proposition in fuzzy production rules for electric locomotive systems

P_1	no current in pantograph	P_{10}	main protective relay breakdown
P_2	traction transformer breakdown	P_{11}	traction power controller breakdown
P_3	pulling motor inoperation	P_{12}	A/D breakdown
P_4	contactor breakdown	P_{13}	25/5V breakdown
P_5	DC750V input breakdown	P_{14}	linear electromotor inoperation
P_6	fan inoperation	P_{15}	phase current sensor inoperation
P_7	main traction inverter breakdown	P_{16}	main circuit system has faults
P_8	DC110V breakdown	P_{17}	power supply system has faults
P_9	control source converter plate breakdown	P_{18}	traction and braking system has faults
P_{19}	electric locomotive systems has faults		

4 Conclusions

In this study, rFRSN P systems are applied in fault diagnosis of electric locomotive systems. This study focuses on offering the causality among breakdown signals, faulty sections, faulty subsystems and faulty electric locomotive systems in the form of fuzzy production rules, processing fault information by syntactical ingredients of rFRSN P systems and proposing fault diagnosis models based on rFRSN P systems for SS4 electric locomotive systems and their subsystems. These models can visually and efficiently describe relationships among breakdown signals detected and candidate faulty sections or faulty systems. This work is an important theoretical basis for proposing a novel bio-inspired method for fault diagnosis of electric locomotive systems by using rFRSN P systems. To test and verify the practical implementation and scalability of the proposed method, our future work include proposing diagnosis algorithms for the models proposed in this paper and model reduction algorithms for the models used in specific cases, and doing plenty of comparison experiments based on a tool to demonstrate its usefulness.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61170016, 61373047, 61170030), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008). The last author acknowledges the support of the project TIN 2012-3734 of the Ministerio de Economía y Competitividad of Spain.

References

1. Gh. Păun, "Computing with membranes," *J. Comput. Syst. Sci.*, 61(1), 108-143 (2000)
2. M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural P systems," *Fund. Inform.*, 71(2-3), 279-308 (2006)
3. Gh. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, "Spike train in spiking neural P systems," *Int. J. Found. Comput. Sci.*, 17(4), 975-1002 (2006)
4. H. Chen, T.-O. Ishdorj, Gh. Păun, and M. J. Pérez-Jiménez, "Handling languages with spiking neural P systems with extended rules," *Romanian J. Inform. Sci. Technol.*, 9(3), 151-162 (2006)
5. R. Freund, M. Ionescu, and M. Oswald, "Extended spiking neural P systems with decaying spikes and/or total spiking," *Int. J. Found. Comput. Sci.*, 19(5), 1223-1234 (2008)
6. M. Cavaliere, O.H. Ibarra, Gh. Păun, O. Egecioglu, M. Ionescu, and S. Woodworth, "Asynchronous spiking neural P systems," *Theor. Comput. Sci.*, 410(24-25), 2352-2364 (2009)
7. L. Q. Pan and Gh. Păun, "Spiking neural P systems: an improved normal form," *Theor. Comput. Sci.*, 411(6), 906-918 (2010)
8. L. Q. Pan and X. X. Zeng, "Small universal spiking neural P systems working in exhaustive mode," *IEEE Trans. on Nanobiosci.*, 10(2), 99-105 (2011)
9. V. P. Metta, K. Krithivasan and D. Garg, "Spiking neural P systems with anti-spikes as transducers," *Rom. J. Inf. Sci. Tech.*, 14(1), 20-30 (2011)
10. M. Ionescu, G. Păun, M. J. Pérez-Jiménez and A. Rodríguez-Patón, "Spiking neural P systems with several types of spikes," *Int. J. Comput. Commun.*, 71(2-3), 648-656 (2011)
11. X. Y. Zhang, B. Luo, X. Y. Fang and L. Q. Pan, "Sequential spiking neural P systems with exhaustive use of rules," *BioSystems*, 108: 52-62 (2012)
12. F. George, C. Cabarle, H. N. Adorna, M. A. Martínez, and M. J. Pérez-Jiménez, "Improving GPU simulations of spiking neural P systems," *Rom. J. Inf. Sci. Tech.*, 15(1), 5-20 (2012)
13. G. C. Francis and N. A. Henry, "On structures and behaviors of spiking neural P systems and Petri nets," *Int. Conf. on Membrane Computing*, pp. 145-160 (2012)
14. T. Song, L. Q. Pan and Gh. Păun, "Asynchronous spiking neural P systems with local synchronization," *Inform. Sciences*, 219, 197-207 (2013)
15. K. Q. Jiang, T. Song and L. Q. Pan, "Universality of sequential spiking neural P systems based on minimum spike number," *Theor. Comput. Sci.*, 499, 88-97 (2013)
16. G. X. Zhang, H. N. Rong, F. Neri and Mario J. Pérez-Jiménez, "An optimization spiking neural P system for approximately solving combinatorial optimization problems," *Int. J. Neural Syst.*, 24(5), 1-15 (2014)
17. J. Wang, P. Shi, H. Peng, Mario J. Pérez-Jiménez, and Tao Wang, "Weighted fuzzy spiking neural P system," *IEEE Trans. Fuzzy Syst.*, 21(2), 209-220 (2013)
18. H. Peng, J. Wang, M. J. Pérez-Jiménez, H. Wang, J. Shao, and T. Wang, "Fuzzy reasoning spiking neural P system for fault diagnosis," *Inform. Sciences*, 235, 106-116 (2013)
19. G. J. Xiong, D. Y. Shi, L. Zhu, and X. Z. Duan, "A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems," *Math. Probl. Eng.*, <http://dx.doi.org/10.1155/2013/815352>, 2013.
20. T. Wang, G. X. Zhang, H. N. Rong, and M. J. Pérez-Jiménez, "Application of fuzzy reasoning spiking neural P systems to fault diagnosis," *Int. J. Comput. Commun.*, 2014. (Accepted)

21. M. Tu, J. Wang, H. Peng, and P. Shi, "Application of adaptive fuzzy spiking neural P systems in fault diagnosis of power systems," *Chinese J. Electron*, 23(1), 87-92 (2014)
22. C. Y. Zhang, "Research on fault diagnosis method of electric locomotive systems based on petri net (M.S Degree Thesis)," Central South University, Changsha, China, 2010.
23. Y. S. Zhang and L. J. Zhu, "Shaoshan4 locomotive," Chinese Railway Publishing House, Beijing, China, 2001.

Graph Cluster Analysis by Lattice based P System with Conditional Rules

Jie Xue and Xiyu Liu

Shandong Normal University, Shandong Jinan 250014, CHINA

Graph clustering is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph, which is widely used in data transformation, network analysis etc [1]. Several algorithms are provided in dealing with graph clustering. However, when the graphs become massive, these algorithms exhibit polynomial or exponential complexity, which make the problem being more challenging. Benefit from the parallelism of P system [2], we present a new P system named lattice based P system with conditional rules(LC P) with the thought of SCAN and k-nearest algorithm for clustering unweighted and weighted graph. Experiment on dolphin social network shows the effectiveness of our algorithm.

The lattice based P system with conditional rules(LC P) is a construct:

$$\Pi = (O, A, C, p, \omega_1, \dots, \omega_m, R^s, R_l, i_0) \quad (1)$$

Where, V is an alphabet; A is a set of variables with real values; C is a set of conditions on rules; P is partial ordered membrane structure of m degree: $(\sigma_1, \sigma_2) \prec \tau$, $(\sigma_1, \sigma_2) \succ \tau$, which is the same as [3]; $\omega_1, \dots, \omega_m$, represents a multiset of strings over $V_i[A_j]$; for a given i and a set of variables from A , with an initial value; i_0 is the output membrane; R^s stands for $(\sigma_1, \sigma_2) \prec \tau$ ordered rules with the following form:

$$\begin{aligned} & [(a[x_a] : c_a, b[x_b] : c_b), up; (e[x_e], f[x_f]), in]^S \\ & \rightarrow [c[x_c], in; d[x_d] : c_d, down | (\sigma_1, \sigma_2) \prec \tau] \\ & [a[x_a], out; b[x_b], in | \sigma_1 \prec \tau], [b[x_b], out; c[x_c], in | \sigma_2 \prec \tau] \\ & \{x_a, x_b, x_c, x_d, x_e, x_f\} \in A, \{a, b, c, d, e, f\} \in V, \{c_a, c_b, c_c, c_d, c_e, c_f\} \in C. \end{aligned}$$

$a[x_a]$ and $b[x_b]$ from σ_1 and σ_2 transform into $c[x_c]$ and go up to τ under the condition c_a and c_b ; simultaneously $d[x_d]$ from τ transforms to $e[x_e]$ into σ_1 and $f[x_f]$ into σ_2 under c_d . If c_a , c_b and c_d are not satisfied, the rule can not execute.

R_l stands for $(\sigma_1, \sigma_2) \succ \tau$ ordered rules with the following form:

$$\begin{aligned} & [(a[x_a] : c_a, b[x_b] : c_b), down; (e[x_e], f[x_f]), in]_I \\ & \rightarrow [c[x_c], in; d[x_d] : c_d, up | (\sigma_1, \sigma_2) \succ \tau] \\ & [a[x_a], out; b[x_b], in | \sigma_1 \succ \tau], [b[x_b], out; c[x_c], in | \sigma_2 \succ \tau] \\ & \{x_a, x_b, x_c, x_d, x_e, x_f\} \in A, \{a, b, c, d, e, f\} \in V, \{c_a, c_b, c_c, c_d, c_e, c_f\} \in C. \end{aligned}$$

$a[x_a]$ and $b[x_b]$ from σ_1 and σ_2 transform into $c[x_c]$ and go down to τ under the condition c_a and c_b ; simultaneously $d[x_d]$ from τ transforms to $e[x_e]$ up to σ_1 and $f[x_f]$ up to σ_2 under c_d . If c_a , c_b and c_d are not satisfied, the rule can not execute.

⁰ Research is supported by the Natural Science Foundation of China(No.61170038), the Natural Science Foundation of Shandong Province(No.ZR2011FM001)

The computation starts with $a[x_a]$ in cells with condition c_a on x_a . The stable, halting status and output of the P system is the same as cell-like P system in maximum.

Structural clustering algorithm for network(SCAN) has good scalability on clustering large graphs [1]. Given an undirected graph, $G = (V, E)$, for a vertex, $\mu \in V$, the neighborhood of μ is $\chi(\mu) = \{\nu | (\mu, \nu) \in E\} \cup \{\mu\}$. The similarity between two vertexes μ and ν is:

$$\delta(\mu, \nu) = \frac{\chi(\mu) \cap \chi(\nu)}{\sqrt{|\chi(\mu)| |\chi(\nu)|}} \quad (2)$$

For a vertex, $\mu \in V$, the ε -neighborhood of μ is $N_\varepsilon(\mu) = \{\nu \in \chi(\mu) | \delta(\mu, \nu) \geq \varepsilon\}$. $\mu \in V$ is a core vertex if $|N_\varepsilon(\mu)| \geq \mu$, where μ is a popularity threshold. SCAN grows clusters from core vertices. If a vertex ν is in the ε -neighborhood of a core μ , then, ν is assign to the same cluster as μ . The lattice based P system with conditional rules for SCAN is constructed as: $O = \{e_{ij}, A_{ij}, N_i, R_{ij}, \alpha, \beta\}$, $1 \leq i \leq n, 1 \leq j \leq n$, $A = \{\delta_{ij}, c_i | 1 \leq i \leq n, 1 \leq j \leq n\}$, $C = \{\delta_{ij}, \delta_{hk} \geq \varepsilon; c_i \geq \theta\}$, $p = \gamma$. $\omega_{\sigma_1} = \omega_{\sigma_2} = \alpha$, $\omega_\tau = e_{ij}[\delta_{ij}]$. In membrane computing, all objects numbers are integer. To adapt it, we amplify δ_{ij} into $\delta_{ij} * 10$.

$R_1 = [e_{ij}[\delta_{ij}], out; \alpha, in | \sigma_1 \succ \tau]$, $R_2 = [e_{hk}[\delta_{hk}], out; \alpha, in | \sigma_2 \succ \tau]$
 $R_{3_1} = [(e_{ij}[\delta_{ij}], e_{hk}[\delta_{hk}] : \delta_{ij}, \delta_{hk} \geq \varepsilon), up; (A_{ij}, A_{hk}), in]^S$
 $\rightarrow [N_i N_j N_h N_k, in; \alpha, down | (\sigma_1, \sigma_2) \succ \tau]$
 $R_{3_2} = [(e_{ij}[\delta_{ij}] : \delta_{ij} \geq \varepsilon, out; A_{ij} N_i N_j, in | \sigma_1 \succ \tau]$
 $R_{4_1} = [(A_{ik}, A_{ij}), up; (\beta, \beta), in]^S \rightarrow [R_{ik} R_{ij}, in; N_i[c_i] : c_i \geq \theta, down | (\sigma_1, \sigma_2) \succ \tau]$
 $R_{4_2} = [(A_{ik}, \lambda), up; (\beta, \beta), in]^S \rightarrow [R_{ik}, in; N_i[c_i] : c_i \geq \theta, down | (\sigma_1, \sigma_2) \succ \tau]$
 $R_{4_3} = [(\lambda, A_{ij}), up; (\beta, \beta), in]^S \rightarrow [R_{ij}, in; N_i[c_i] : c_i \geq \theta, down | (\sigma_1, \sigma_2) \succ \tau]$

Membrane structure is $(\sigma_1, \sigma_2) \succ \tau$. At the first step, two copies of $\{e_{ij}[\delta_{ij}]\}$ are sent into σ_1 and σ_2 by R_1 and R_2 . α is sent to τ simultaneously. If the similarity of two points is larger than ε , they are neighbors. R_3 changes neighbors into A_{ij} and produce N_i . All edges will go through rule R_3 and produce multiple numbers of N_i . When the number of N_i arrives at θ , R_4 is triggered. i is the core points of a cluster, all A with i as the first subscript are set as a member of cluster i and change into R_{ij} .

SCAN algorithm takes n steps to complete the computation at least. Whereas, the whole process is completed in $n/2 + d_{max}/2$ of SCAN designed by LC P system. k-nearest based algorithm by LC P system is researched as well. Time complexity is within $[\frac{k}{2} + 1, k + 1]$. Traditional algorithm consumes $O(n)$, $k \leq n$. LC P system based SCAN and k-nearest algorithms reduce time complexity.

References

- [1] Han, Jiawei and Kamber, Micheline, *Data Mining, Third Edition: Concepts and Techniques*, Elsevier, 2012.
- [2] Păun G, Rozenberg G, and Salomaa A., *Membrane Computing*, Oxford University Press, New York, 2010.
- [3] Jie Xue, Xiyu Liu, *Lattice Based Communication P Systems with Applications in Cluster Analysis*, Soft Computing, 2013:1-16, DOI 10.1007/s00500-013-1155-y.

Author Index

A	
Adorna, H.	153
Ahmed, T.	25, 345
Alhazov, A.	41, 61, 71, 99, 117, 333
Aman, B.	41, 129
B	
Bakir, M.B.	135
Burtseva, L.	333
C	
Cabarle, F.G.	153
Cienciala, L.	3, 155
Ciencialová, L.	155
Cojocarui, S.	333
Colesnicov, A.	333
Csajbók, Z.E.	283
Csuhaj-Varjú, E.	7, 129
F	
Freund, R.	41, 61, 71, 99, 117, 129
G	
García-Quismondo, M.	169
Gazdag, Z.	191, 205
Gheorghe, M.	7
Grützmann, K.	217
Gutiérrez-Naranjo, M.A.	191, 205
H	
Höckner, B.	217
Hayat, S.	217
Hinze, T.	217
I	
Ipate, F.	135
Ivanov, S.	99, 235
K	
Kolonits, G.	205
Konur, S.	135
Krithivasan, K.	267
L	
Langer, M.	155
Leporati, A.	19, 251
M	
Malahov, L.	333
Manzoni, L.	19, 251
Martínez-Del-Amor, M.A.	169
Mauri, G.	19, 251
Metta, V.P.	267
Mierla, L.	135
Mihálydeák, T.	283
N	
Nicolescu, R.	287
Niculescu, I.	135
O	
Obtulowicz, A.	313
P	
Pérez-Jiménez, M.J.	11, 169, 361
Perdek, M.	155
Porreca, A.E.	19, 251
R	
Raghuraman, S.	267
Rogozhin, Y.	333
S	
Sauer, P.	217
Srivastava, A.	25, 345
V	
Vaszil, G.	7
Verlan, S.	117
Verma, R.	25, 345
W	
Wang, T.	361
Wiedermann, J.	15
X	
Xue, J.	375
Z	
Zandron, C.	19, 251
Zhang, G.	361