Marian Gheorghe Thomas Hinze
Gheorghe Păun Grzegorz Rozenberg
Arto Salomaa (Eds.)

LNCS 6501

Membrane Computing

11th International Conference, CMC 2010
Jena, Germany, August 2010
Revised Selected Papers

Springer

Lecture Notes in Computer Science 6501

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Marian Gheorghe Thomas Hinze
Gheorghe Păun Grzegorz Rozenberg
Arto Salomaa (Eds.)

Membrane Computing

11th International Conference, CMC 2010
Jena, Germany, August 24-27, 2010
Revised Selected Papers

Springer

Volume Editors

Marian Gheorghe
University of Sheffield, Department of Computer Science
Regent Court, Portobello Street, Sheffield S1 4DP, UK
E-mail: m.gheorghe@dcs.shef.ac.uk

Thomas Hinze
Friedrich Schiller University, Department of Bioinformatics
School of Biology and Pharmacy
Ernst-Abbe-Platz 1–4, 07743, Jena, Germany
E-mail: thomas.hinze@uni-jena.de

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
P.O. Box 1-764, 014700 Bucharest, Romania
E-mail: george.paun@imar.ro; gpaun@us.es

Grzegorz Rozenberg
Leiden University, Leiden Center of Advanced Computer Science (LIACS)
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
E-mail: rozenber@liacs.nl

Arto Salomaa
Turku Centre for Computer Science (TUCS)
Leminkäisenkatu 14, 20520 Turku, Finland
E-mail: asalomaa@cs.utu.fi

Library of Congress Control Number: 2010941767

CR Subject Classification (1998): F.1, F.4, I.6, J.3, C.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-18122-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-18122-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume contains a selection of papers presented at the 11th International Conference on Membrane Computing (CMC11, `http://cmc11.uni-jena.de`) which took place in Jena, Germany, during August 24–27, 2010.

The first three workshops on membrane computing were organized in Curtea de Argeş, Romania – they took place in August 2000 (with the proceedings published in *Lecture Notes in Computer Science*, volume 2235), in August 2001 (with a selection of papers published as a special issue of *Fundamenta Informaticae*, volume 49, numbers 1–3, 2002), and in August 2002 (with the proceedings published in *Lecture Notes in Computer Science*, volume 2597). 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*, by Springer, as volumes 2933, 3365, 3850, 4361, 4860, and 5391, respectively. The 10th workshop returned to Curtea de Argeş in August 2009 (LNCS volume 5957).

From then on, the workshop became a conference and the series of meetings on membrane computing continues as the Conference on Membrane Computing, with the 2010 edition, CMC11, held in Jena, Germany.

The invited speakers for CMC11 were: Gabriel Ciobanu (Iasi, Romania), Peter Dittrich (Jena, Germany), Marian Gheorghe (Sheffield, UK), Martin Kutrib (Gießen, Germany), Maurice Margenstern (Metz, France), and Gheorghe Păun (Bucharest, Romania, and Seville, Spain). Extended abstracts of these talks are included in this volume. Moreover, the CMC11 and Jena Life Science Forum 2010 (JLSF2010) audiences enjoyed the opportunity to listen to two joint keynote presentations delivered by Gheorghe Păun (Bucharest, Romania and Seville, Spain), from CMC11, and Peter Stadler (Leipzig, Germany), from JLSF2010.

This volume also incorporates a selection of 23 accepted papers. Each of them was subject of at least three referee reports. The Program Committee consisted of 21 members: Artiom Alhazov (Hiroshima, Japan), Gabriel Ciobanu (Iasi, Romania), Erzsebet Csuhaj-Varju (Budapest, Hungary), Gabi Escuela (Jena, Germany), Rudolf Freund (Vienna, Austria), Pierluigi Frisco (Edinburgh, UK), Marian Gheorghe (Sheffield, UK) – Chair, Thomas Hinze (Jena, Germany) – Co-chair, Oscar H. Ibarra (Santa Barbara, USA), Vincenzo Manca (Verona, Italy), Maurice Margenstern (Metz, France), Giancarlo Mauri (Milan, Italy), Van Nguyen (Adelaide, Australia), Marion Oswald (Budapest, Hungary), Linqiang Pan (Wuhan, China), Gheorghe Păun (Bucharest, Romania and Seville, Spain), Mario J. Perez-Jimenez (Seville, Spain), Dario Pescini (Milan, Italy), Francisco J. Romero-Campero (Nottingham, UK), Monika Sturm (Dresden, Germany), and Sergey Verlan (Paris, France). It was assisted in the selection

process by seven additional reviewers: Oana Agrigoroaiei (Iasi, Romania), Christian Bodenstein (Jena, Germany), Paolo Cazzaniga (Milan, Italy), Alberto Leporati (Milan, Italy), Antonio E. Porreca (Milan, Italy), Sara Woodworth (Amgen in Thousand Oaks, USA), and Claudio Zandron (Milan, Italy).

The Organizing Committee was constituted by Jörn Behre, Gabi Escuela, Thomas Hinze – Chair, Thorsten Lenser, and Kathrin Schowtka (Secretary).

Since the meeting became a conference, its structure was modified quite considerably. First, three international satellite workshops were organized: The 4th Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC), the Second Workshop on Non-Classical Models of Automata and Applications (NCMA), and the new Workshop on Applications of Membrane Computing, Concurrency and Agent-Based Modeling in Population Biology (AMCA-POP). Second, a software demo session was included in the program, with the intention of becoming a platform allowing participants to share computer programs and tools applicable and useful in the field of membrane computing. Third, a poster session took place, providing an opportunity of attracting presentations of late-breaking results from young researchers and students new in the field. Finally, the Best Contribution Award, consisting of a travel grant, was given for the first time. Its recipient, identified by the vote of all CMC11 participants, was Pierluigi Frisco (Edinburgh, UK).

We gratefully acknowledge funding for CMC11 from the German Research Foundation (grant HI801/3-1), and additional financial support provided by the Jena Centre for Bioinformatics (JCB). Furthermore, we thank the administration of the Friedrich Schiller University Jena for the perfect infrastructure made available to CMC11, and for the extensive assistance in many issues related to CMC11. Finally, we express our gratitude to the "UniverCity" of Jena for providing special rates for accomodation to CMC11 participants.

The work of G. Păun in editing this volume was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

The editors warmly thank the Program Committee, the invited speakers, the authors of the submitted papers, the reviewers, and all the participants. Special thanks are due to Springer for the pleasant cooperation in the timely production of this volume.

October 2010

Marian Gheorghe
Thomas Hinze
Gheorghe Păun
Grzegorz Rozenberg
Arto Salomaa

Table of Contents

Keynote Presentations

Invited Presentations

Regular Presentations

Membrane Computing at Twelve Years

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
P.O. Box 1-764, 014700 Bucureşti, Romania, and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Abstract. This is not a balance sheet of the domain, although twelve years is a good time for a research area to have an overall assessment, a comparison of initial dreams with the achievements, an overview of main results, an estimation of research directions for the near future. This would be too an ambitious task, although an useful exercise, so I am postponing it. What I want however to mention is that the initial aims were rather restricted, confined to computing theory, not imaging, for instance, that this area can become a framework for modeling, mainly of biological processes, but also for other areas. "There is nothing more practical than a good theory" one often says. Membrane computing proved to be practical; whether or not this means that it provides a *good* theory, let us assess it after twelve more years.

The presentation here has two main aims.

First, it tries to introduce membrane computing through twelve basic ideas about which I have worked along the years: cell-like P systems, string objects, symport-antiport rules (computing by communication), active membranes (especially, membrane division and membrane creation), tissue-like P systems, using P systems in the accepting mode, trace languages, numerical P systems, P systems with objects on membranes (brane calculi inspired P systems), P colonies, spiking neural P systems, dP systems. Other topics will be touched, having or not connections with my work: controls on rule application, minimal parallelism, asynchronous systems, array objects, trees as a result of computations, population P systems, conformon objects, probabilistic-stochastic P systems, MP systems, complexity approaches, and so on. Also, some open problems and research topics will be mentioned.

Second, some hints will be given concerning applications, especially in modeling biological processes, but also in studying-simulating ecosystems; applications in economics, computer graphics, approximate optimization, etc., will be also mentioned.

Very few precise bibliographical information will be given. Instead, the reader will be refereed to the existing books available – see below – and to the website of membrane computing, from `http://ppage.psystems.eu`. A lot of information can also be found in the proceedings of the two yearly meetings in the area, Brainstorming Week on Membrane Computing and Workshop on Membrane Computing (the latter one being

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 1–2, 2010.

the ancestor of the present conference), in the PhD theses with membrane computing subject, and in the special issues of journals devoted to membrane computing; details about all these can be found in the above mentioned website.

References

1. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Berlin (2006)
2. Frisco, P.: Computing with Cells. Advances in Membrane Computing. Oxford Univ. Press, Oxford (2009)
3. Păun, A.: Computability of the DNA and Cells. Splicing and Membrane Computing. SBEB Publ., Choudrant (2008)
4. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
5. Păn, G., Rozenberg, A., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press, Oxford (2010)

Testing Based on P Systems – An Overview

Marian Gheorghe[1,2] and Florentin Ipate[2]

[1] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk
[2] Department of Computer Science, Faculty of Mathematics and Computer Science
The University of Pitesti
Str Targu din Vale 1, 110040 Pitesti
florentin.ipate@ifsoft.ro

Abstract. In this extended abstract there are surveyed various testing approaches utilised so far for applications based on P systems.

1 Introduction

All software applications, as any engineering products, irrespective of their nature and purpose, are thoroughly tested before being released, installed and used. Testing appears everywhere, is part of any technology, and does not have a substitute. In many hardware or software systems testing is conducted together with formal verification, especially when a certain formal model is utilised. In software industry testing is a necessary mechanism to increase the confidence in the product correctness and to make sure it works properly.

P systems area, initially introduced by [11], has been under an intensive investigation in the last decade. It covers a broad range of aspects, from theoretical investigations of the computational power and descriptional complexity of various mechanisms, to applications in modelling different natural or engineered systems, and from interactions with other computational models to implementations of various problems utilising either certain tools or general purpose programming languages. An account of the various developments of the field, mostly at the theoretical level, is provided in [13], [12]; applications of P systems are presented in [2]. The most recent research aspects of this field are reported in [14].

In this note it will be overviewed an approach on black box testing which requires that for a given specification defined as a P system, an implementation of it exists and this will be tested utilising a test set derived from its specification.

Testing P systems has been so far considered by using certain coverage principles. More often the rule coverage is utilised, by taking into account different contexts. In order to reveal the usage of the rules, grammar and automaton based methods are derived from P systems specifications. These two types of testing methods are reviewed in the following sections. Some specific test set generation methods are analysed and discussed.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 3–6, 2010.

2 Grammar Based Methods

In order to test an implementation developed from a P system specification, a test set is built, in a black box manner, as a finite set of sequences containing references to rules.

Although there are similarities between context-free grammars utilised in grammar testing and basic P systems, that we aim to consider, there are also major differences that pose new problems in defining testing methods and strategies to obtain tests sets. Some of the difficulties encountered when some grammar-like testing procedures are introduced, are related to: the hierarchical compartmentalisation of the P system model, parallel behaviour, communication mechanisms, the lack of a non-terminal alphabet and the use of multisets of objects instead of sets of strings.

The rule coverage criteria discussed will be illustrated for one compartment P system, i.e., $\Pi = (V, \mu, w, R)$, where $\mu = [_1]_1$. The simplest and most basic rule coverage criterion, called *rule coverage*, is defined in such a way that every rule from R is covered by a certain computation; i.e., for each rule $r \in R, r : a \rightarrow v$, there is a multiset u_r over V which *covers* r (there is a computation $w \Longrightarrow^* xay \Longrightarrow x'vy' \Longrightarrow^* u_r$; $w, x, y, v, u_r \in V^*$, $a \in V$). Some more complex coverage criteria can be considered (see [4], [5]).

Let us consider the following one compartment P system, $\Pi_1 = (V_1, \mu_1, w_1, R_1)$, where $V_1 = \{s, a, b, c\}$; $\mu_1 = [_1]_1$ - i.e., one compartment, denoted by 1; $w_1 = s$; $R_1 = \{r_1 : s \rightarrow ab, r_2 : a \rightarrow c, r_3 : b \rightarrow bc, r_4 : b \rightarrow c\}$. Each multiset w, will be denoted by a vector of non-negative integer numbers $(|w|_s, |w|_a, |w|_b, |w|_c)$. Test sets for Π_1 satisfying the rule coverage criterion are

- $T_{1,1} = \{(0,1,1,0), (0,0,1,2), (0,0,0,2)\}$ and
- $T_{1,2} = \{(0,1,1,0), (0,0,1,2), (0,0,0,3)\}$.

More complex test sets can be built with the same elements, by considering sequences of multisets rather than simple multisets as above [5].

3 Finite State Machine Based Methods

Finite state machine based testing is widely used for software testing. It provides very efficient and exhaustive testing strategies and well investigated methods to generate test sets. In this case it is assumed that a model of the system under test is provided in the form of a finite state machine. In our case we will consider a way to obtain such a machine from a partial computation in a P system. More precisely we will consider computations of at most k steps, for a given integer k, starting from the initial multisets. These can be considered paths in an automaton defining partial computations of no more than k steps. The minimal automaton covering it can be now utilised as a model to generate test sets (see [5], [7]).

A different aproach can be also considered by using a special class of state machines, called X-machines. Given that the relationships between various classes

of P systems and these machines are well studied ([14], [1]) and the X-machine based testing is well developed, standard techniques for generatig test sets based on X-machines can be adapted to the case of P systems [6].

Specific coverage criteria can be defined in the case of finite state machine based testing. One such criterion, called *transition coverage*, aims to produce a test set in such a way that every single transition of the model is covered.

If we build a finite state machine associated with the previous P system, Π_1, for partial computations of length at most 4, then a test set satisfying the transition cover principle is
$T_{1,s} = \{(1,0,0,0),(0,1,1,0),(0,0,1,2),(0,0,0,2),(0,0,1,3),(0,0,0,3)\}$.

The transition cover criterion, however, does not only depend on the rules applied, but also on the state reached by the system when a given rule has been applied.

4 Generating Test Sets Using Model Checking

The generation of different test sets, according to certain coverage criteria, can be done by utilising some specific algorithms or by applying some tools that indirectly will generate test sets. Such tools, like model checkers, can be used to verify some general properties of a model and when these are not fulfilled then some counter-examples are produced, which act as test sets in certain circumstances.

In the case of P systems an encoding based on a Kripke structure associated with the system is provided for model checkers like NuSMV [9] or SPIN [10]. This relies on certain operations defined in [3] and encapsulates the main features of a P system, including maximal parallelism and communication, but within a finite space of values associated with the objects present in the system. The rule coverage principle is expressed by using temporal logics queries available in such contexts. By negating specific coverage criteria, counter-examples are generated. For instance the rule coverage set $T_{1,1}$ can be obtained in this way.

5 Conclusions

P systems based testing methods are reviewed and some coverage principles presented. Two main classes of methods, based on grammars and finite state machines, are introduced and specific test generation tools based on model checking techniques are mentioned. Apart from these methods some other approaches have been considered when mutation techniques have been employed [8].

The main goal of this research will be to develop specific testing methods that make direct use of the concepts and elements, both static and dynamic, of P systems and identify specific coverage principles, like compartment coverage. These approaches should provide techniques to reveal the effect of very complex interactions between the many components of such models.

Acknowledgements. The research of MG and FI is supported by CNCSIS - UEFISCSU, project no.643/2008, *An integrated evolutionary approach to formal modelling and testing.*

References

1. Aguado, J., Bălănescu, T., Cowling, A., Gheorghe, M., Holcombe, M., Ipate, F.: P systems with replicated rewriting and stream X-machines (Eilenberg machines). Fundamenta Informaticae 49, 17–33 (2002)
2. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): Applications of membrane computing. Natural Computing Series. Springer, Heidelberg (2006)
3. Dang, Z., Ibarra, O.H., Li, C., Xie, G.: Decidability of model-checking P systems. Journal of Automata, Languages and Combinatorics 11, 179–198 (2006)
4. Gheorghe, M., Ipate, F.: On testing P systems. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 204–216. Springer, Heidelberg (2009)
5. Gheorghe, M., Ipate, F., Lefticaru, R., Dragomir, C.: An integrated approach to P systems formal verification. In: Proceedings of the 11th Conference on Membrane Computing, Jena, Germany, pp. 225–238 (2010)
6. Ipate, F., Gheorghe, M.: Testing non-deterministic stream X-machine models and P systems. Electronic Notes in Theoretical Computer Science 227, 113–226 (2008)
7. Ipate, F., Gheorghe, M.: Finite state based testing of P systems. Natural Computing 8, 833–846 (2009)
8. Ipate, F., Gheorghe, M.: Mutation based testing of P systems. International Journal of Computers, Communications & Control 4, 253–262 (2009)
9. Ipate, F., Gheorghe, M., Lefticaru, R.: Test generation from P systems using model checking. Journal of Logic and Algebraic Programming 79, 350–362 (2010)
10. Ipate, F., Lefticaru, R., Tudose, C.: Formal verification of P systems using SPIN. International Journal of Foundations of Computer Science (2010) (submitted)
11. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000)
12. Păun, G.: Membrane computing. An introduction. Springer, Berlin (2002)
13. Păun, G., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science 287, 73–100 (2002)
14. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford handbook of membrane computing. The Oxford University Press, Oxford (2009)

Mobility in Computer Science and in Membrane Systems

Gabriel Ciobanu

Romanian, Academy, Institute of Computer Science
and "A.I.Cuza" University of Iaşi
Blvd. Carol I no.11, 700506 Iaşi, Romania
gabriel@iit.tuiasi.ro, gabriel@info.uaic.ro

In Memory of Robin Milner (1934-2010)

Mathematical models are useful in different fields to provide a deeper and more insightful understanding of various systems and notions. We refer here to the formal description of *mobility* in computer science. The first formalism in computer science able to describe mobility is the π-calculus [16]. It was followed by ambient calculus [6]. A biologically-inspired version of ambient calculus is given by bioambients [19] and several brane calculi [5]. On the other hand, systems of mobile membranes [13] represent other formalisms with mobility in the framework of membrane computing.

1 Mobility in Process Calculi

When expressing mobility, we should mention what entities move and in what space they move. There are several possibilities: processes moving in a physical space of computing locations, processes moving in a virtual space of linked processes, links moving in a virtual space of linked processes, etc. The π-calculus is a formalism where links are the moving entities, and they move in a virtual space of linked processes (the network of web pages is a good example for this approach). This option is powerful enough to express moving processes both in a physical space of computing locations and in a virtual space of linked processes [16].

The π-calculus was developed as a calculus of communicating systems that allows the representation of concurrent computations whose configuration may change during the computation. The computational world of the π-calculus contains just processes (also called agents) and channels (also called ports). In contrast to λ-calculus which represents computations through functions, the π-calculus uses the process as an abstraction of an independent thread of control. A channel is an abstraction of the communication link between processes, and processes interact by sending information through these channels. Since variables may be channel names, computation can change the channel topology and process mobility is supported. Milner emphasized the importance of identifying the "elements of interaction" [15], and his π-calculus extends the Church-Turing model by adding the interaction between a sender and a receiver to the algebraic

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 7–17, 2010.

elegance of λ-calculus. The π-calculus has a simple semantics and a tractable algebraic theory [16]. Actually the π-calculus is a widely accepted model of interacting systems with dynamically evolving communication topology and (channel) mobility. Its mobility increases the expressive power enabling the description of many high-level concurrent features. The π-calculus can model networks in which messages are sent from one site to another site and may contains links to active processes or to other sites; it is a general model of computation which takes interaction as primitive.

We shortly present the monadic version of the π-calculus ("monadic" means that the messages sent between processes consist of exactly one name). Let $\mathcal{X}$ be a infinite countable set of *names*. The elements of $\mathcal{X}$ are denoted by $x, y, z, \ldots$ The terms (expressions) of this formalism are called processes, and they are denoted by $P, Q, R, \ldots$.

The **processes** are defined over the set $\mathcal{X}$ of names by the following grammar

$$P \ ::= \ 0 \ \mid \ \overline{x}\langle z\rangle.P \ \mid \ x(y).P \ \mid \ P \mid Q \ \mid \ P + Q \ \mid \ !P \ \mid \ \nu x \ P$$

The π-calculus expressions are defined by guarded processes $\overline{x}\langle z\rangle.P$ and $x(y).P$, parallel composition $P \mid Q$, nondeterministic choice $P + Q$, replication $!P$ and a restriction $\nu x \ P$ creating a local fresh channel x for process P. π-calculus replication $!P$ can also be expressed by recursive equations of parametric processes. 0 is the empty process.

Input guards and output guards represent sending and receiving a channel name along a link. The output guarded process $\overline{x}\langle z\rangle.P$ sends z along x and then, after the output has completed, continues as P. An input guarded process $x(y).Q$ waits until a name is received along x, substitutes it for the bound variable y and continues as Q [1]. The parallel composition $\overline{x}\langle z\rangle.P \mid x(y).Q$ may thus synchronize on x, and so the processes can interact by using channels they share. A name received in one interaction can be used in another; by receiving a channel name, a process can interact with processes which are unknown to it, but now they share the same channel name. This aspect is important in defining the π-calculus mobility, together with the scope of names defined by $\nu x \ P$ and extrusion of names from their scopes.

Over the set of processes it is defined a structural congruence relation $\equiv$ providing a static semantics. The structural congruence is defined as the smallest congruence over the set of processes which satisfies the following equalities involving the set $fn(P)$ of the free occurrences in a process P and standard α-conversion denoted by $=_\alpha$:

– $P \equiv Q$ if $P =_\alpha Q$

[1] There is an important distinction between input and output guards. Output guard is a simple sending of a name z along a channel x, but the input guard has a more complex action: the name received along the channel x replaces y in the process following the input guard. Input guard is a *binding* operator involving substitutions: in $x(y).P$, the name y binds free occurrences of y in P. In a second binding operator $\nu x \ P$, the name x binds free occurrences of x in P.

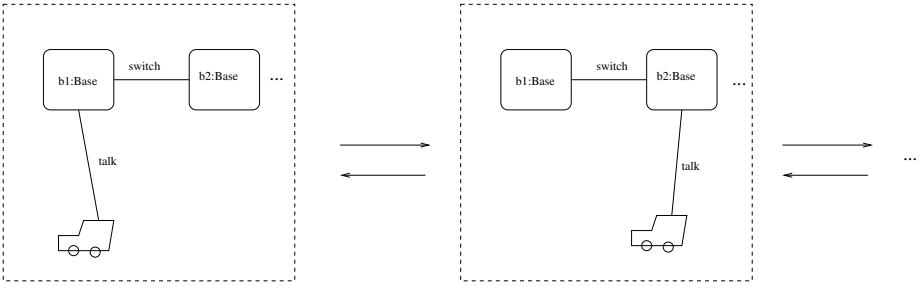
- $P + 0 \equiv P$, $P + Q \equiv Q + P$, $(P + Q) + R \equiv P + (Q + R)$,
- $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$,
- $!P \equiv P \mid !P$
- $\nu x 0 \equiv 0$, $\nu x \nu y P \equiv \nu y \nu x P$, $\nu x(P \mid Q) \equiv P \mid \nu x Q$ if $x \notin fn(P)$.

The rule $\nu x(P \mid Q) \equiv P \mid \nu x Q$ whenever $x \notin fn(P)$ describes the extrusion of names from their scope, and it plays an important role in defining the π-calculus mobility.

The evolution of a process is described in π-calculus by a reduction relation over processes called *reaction*. This relation contains those transitions which can be inferred from a set of rules. The reduction relation over processes is defined as the smallest relation $\rightarrow$ satisfying the following rules:

(com) $(\overline{x}\langle z\rangle.P + R_1) \mid (\, x(y).Q + R_2) \rightarrow P \mid Q\{z/y\}$
(par) $P \rightarrow Q$ implies $P \mid R \rightarrow Q \mid R$
(res) $P \rightarrow Q$ implies $(\nu x)P \rightarrow (\nu x)Q$
(str) $P \equiv P'$, $P' \rightarrow Q'$ and $Q' \equiv Q$ implies $P \rightarrow Q$

We give an example describing a simple interaction between a handphone carried in a car and two base stations. The connections between the car (handphone) and a base station can change as the car is moving around.



We consider three processes B_1, B_2 and C corresponding to the two base stations and the car, respectively. We start with their parallel composition $B_1 \mid C \mid B_2$ described by the left square of the picture. The base B_1 and the car C are connected by a channel *talk*, and B_1 and B_2 by a channel *switch*. This means that *talk* is free in both B_1 and C, and *switch* is a free name in both B_1 and B_2. By the process expression $\nu\ talk\ (B_1 \mid C) \mid B_2$, the name *talk* is restricted to B_1 and C, and we interpret that B_1 and C have an exclusive communication along the channel *talk*. If $B_1 = \overline{switch}\langle talk\rangle.B_1'$, then base B_1 wishes to send the name of channel *talk* to base B_2 along the channel *switch*. Moreover, if *talk* is not free in B_1' ($talk \notin fn(B_1')$), then B_1' will lose its link to C. Base B_2 is waiting for a channel name sent by B_1, namely $B_2 = switch(y).B_2'$. Applying the reduction rule (com) and the extrusion of name *talk* from its previous scope given by B, we get the transition

$$\nu\ talk\ (B_1 \mid C) \mid B_2 \ \longrightarrow\ B_1' \mid \nu\ talk\ (C \mid B_2'')$$

where $B_2'' = B_2'\{talk/y\}$. The initial process $\nu\ talk\ (B_1 \mid C) \mid B_2$ changes its communication topology, and it becomes as it is described in the right square of the figure above. Now B_2'' and C have an exclusive communication along the channel $talk$. This is essentially the mobility mechanism offered by the π-calculus. More details are in [16].

Various forms of behavioural equivalence in process algebras are based on the notion of bisimulation. There are several definitions in the literature for bisimulation; their definitions are given by using the labeled transition system defined by the reduction rules. Systems can be checked automatically by studying the bisimilarity between two processes, namely the model and its specification. The properties of finite state transition systems can be specified in a very powerful logic called μ-calculus. Thus it is possible to use various verification techniques for proving properties about the mobile concurrent systems modelled in the π-calculus. Modelling with π-calculus and verifying with μ-calculus and some of its proper subsets have been thoroughly investigated in the literature. Model checking π-calculus processes is discussed in several papers, and Mobility Workbench [21] is the software tool supporting this model checking.

There are several variants and extensions of the π-calculus: Spi, Dpi, tDpi, appliedPi, bigraphs. An important change is introduced in the distributed version Dpi of the π-calculus presented in [11]: the mobility is expressed in a simpler way, by using an explicit migration primitive $goto\ l.P$ enabling mobility between explicit locations names. A timed distributed π-calculus is introduced and study in [8]. Regev and Shapiro use the π-calculus in describing the biochemical systems by abstracting "cell-as-computation", and using processes as abstractions of molecules in biomolecular systems [20]; the authors use these abstractions for representation, simulation, and analysis of metabolic pathways.

Another formalism able to express mobility is *ambient calculus* [6]. Ambient calculus describe computation carried out on mobile devices (i.e. networks having a dynamic topology), and mobile computation (i.e. executable code able to move around the network). The primitive concept of the ambient calculus is the ambient defined as a bounded place in which computation can occur. The ambients can be nested inside other ambients. Each ambient has a name used to control the access to this ambient. Computation is represented as the movement of ambients: they can be moved as a whole, changing their location by consuming certain capabilities: in, out, open. These basic operations are expressive enough to simulate name-passing channels in the π-calculus. In certain conditions, the π-calculus is also able to simulate the ambient calculus [10].

We consider a variant of mobile ambients called safe ambients for which the movement of an ambient takes place only if both participants agree [14]. The mobility is provided by the consumption of *pairs* of capabilities. The safe ambients differ from ambients by co-actions: if in ambients a movement is initiated only by the moving ambient and the target ambient has no control over it, in safe ambients both participants must agree by using a matching between an action and its co-action. We present here a short description of pure safe ambients (SA); more information can be found in [14].

Given an infinite set of names $\mathcal{N}$ (ranged over by $m, n, \ldots$), we define the set $\mathcal{A}$ of SA-processes (denoted by $A, A', B, B', \ldots$) together with their capabilities (denoted by $C, C', \ldots$) as follows:

$$C ::= in\ n \mid \overline{in}\ n \mid out\ n \mid \overline{out}\ n \mid open\ n \mid \overline{open}\ n$$
$$A ::= \mathbf{0} \mid C.A \mid n[\ A\ ] \mid A \mid B$$

Process $\mathbf{0}$ is an inactive mobile ambient. A movement $C.A$ is provided by the capability C, followed by the execution of A. An ambient $n[\ A\ ]$ represents a bounded place labelled by n in which a SA-process A is executed. $A \mid B$ is a parallel composition of mobile ambients A and B.

The *structural congruence* $\equiv_{amb}$ over ambients is the least congruence such that $(\mathcal{A}, |, \mathbf{0})$ is a commutative monoid. The operational semantics of safe ambients is given in terms of a reduction relation $\Rightarrow_{amb}$ by the following axioms and rules:

Axioms

$$(In)\ \ n[\ in\ m.A \mid A'\ ] \mid m[\ \overline{in}\ m.B \mid B'\ ] \Rightarrow_{amb} m[\ n[\ A \mid A'\ ] \mid B \mid B'\ ];$$
$$(Out)\ \ m[\ n[\ out\ m.A \mid A'\ ] \mid \overline{out}\ m.B \mid B'\ ] \Rightarrow_{amb} n[\ A \mid A'\ ] \mid m[\ B \mid B'\ ];$$
$$(Open)\ \ open\ n.A \mid n[\ \overline{open}\ n.B \mid B'\ ] \Rightarrow_{amb} A \mid B \mid B'\ .$$

Rules:

$$(Comp1)\ \frac{A \Rightarrow_{amb} A'}{A \mid B \Rightarrow_{amb} A' \mid B};\quad (Comp2)\ \frac{A \Rightarrow_{amb} A' \quad B \Rightarrow_{amb} B'}{A \mid B \Rightarrow_{amb} A' \mid B'};$$
$$(Amb)\ \ \frac{A \Rightarrow_{amb} A'}{n[\ A\ ] \Rightarrow_{amb} n[\ A'\ ]}\ ;\quad (Struc)\ \frac{A \equiv A',\ A' \Rightarrow_{amb} B',\ B' \equiv B}{A \Rightarrow_{amb} B}.$$

$\Rightarrow^*_{amb}$ denotes a reflexive and transitive closure of the binary relation $\Rightarrow_{amb}$.

Mobile ambients are well suited to express various aspects of mobile computations like working environment and access to resources. On the other hand, bioambients are introduced as abstraction for biological compartments [19].

The biological inspiration is predominant in the case of *brane calculus* [5]. The operations of the two basic brane calculi, namely *pino, exo, phago* (for the PEP fragment) and *mate, drip, bud* (for the MBD fragment of brane calculus) are directly inspired by the biologic processes of *endocytosis, exocytosis* and *mitosis*. Since some proteins are embedded in cell membranes, and can act on both sides of the membrane simultaneously, brane calculus use both sides of the membrane, emphasizing that computation happens also on the membrane surface. We present here an overview of PEP fragment of brane calculus without replication. Cardelli motivates that the replication operator is used to model the notion of a "multitude" of components of the same kind, which is in fact a standard situation in biology. We do not consider the replicator operator because we are not able to define a corresponding membrane system without knowing exactly the initial membrane structure. More details on brane calculus can be found in [5].

A membrane structure consists of a collection of nested membranes as can be seen from Table 1. Membranes are formed of patches σ, where a patch can be composed from other patches $\sigma \mid \tau$. A patch σ consists of an action a followed, after its consumption, by another patch σ_1: $\sigma = a.\sigma_1$. Actions often come

Table 1. Pino/Exo/Phage Calculus Syntax

Systems	$P,Q ::= P \circ Q \mid \sigma(\,) \mid \sigma(P)$	nests of membranes
Branes	$\sigma,\tau ::= O \mid \sigma \mid \tau \mid a.\sigma$	combinations of actions
Actions	$a,b ::= n^{\searrow} \mid \overline{n}^{\searrow}(\sigma) \mid n^{\nwarrow} \mid \overline{n}^{\nwarrow} \mid pino(\sigma)$	phago $\searrow$, exo $\nwarrow$

in complementary pairs which cause the interaction between membranes. The names n are used to pair-up actions and co-actions.

We abbreviate $a.0$ as a, $0(P)$ as (P), and $0(\,)$ as $(\,)$. The structural congruence relation is a way of rearranging the system such that the interacting parts come together; the structural congruence $\equiv_b$ is defined in Table 2.

Table 2. Pino/Exo/Phage Structural Congruence

$P \circ Q \equiv_b Q \circ P$	$\sigma \mid \tau \equiv_b \tau \mid \sigma$
$P \circ (Q \circ R) \equiv_b (P \circ Q) \circ R$	$\sigma \mid (\tau \mid \rho) \equiv_b (\sigma \mid \tau) \mid \rho$
	$\sigma \mid 0 \equiv_b \sigma$
$P \equiv_b Q$ implies $P \circ R \equiv_b Q \circ R$	$\sigma \equiv_b \tau$ implies $\sigma \mid \rho \equiv_b \tau \mid \rho$
$P \equiv_b Q$ and $\sigma \equiv_b \tau$ implies $\sigma(P) \equiv_b \tau(Q)$	$\sigma \equiv_b \tau$ implies $a.\sigma \equiv_b a.\tau$

Table 3. Pino/Exo/Phago Calculus Reduction Rules

$pino(\rho).\sigma \mid \sigma_0(P) \rightarrow_b \sigma \mid \sigma_0(\rho(\,) \circ P)$	Pino
$\overline{n}^{\nwarrow}.\tau \mid \tau_0(n^{\nwarrow}.\sigma \mid \sigma_0(P) \circ Q) \rightarrow_b P \circ \sigma \mid \sigma_0 \mid \tau \mid \tau_0(Q)$	Exo
$n^{\searrow}.\sigma \mid \sigma_0(P) \circ \overline{n}^{\searrow}(\rho).\tau \mid \tau_0(Q) \rightarrow_b \tau \mid \tau_0(\rho(\sigma \mid \sigma_0(P)) \circ Q)$	Phago
$P \rightarrow_b Q$ implies $P \circ R \rightarrow_b Q \circ R$	Par
$P \rightarrow_b Q$ implies $\sigma(P) \rightarrow_b \sigma(Q)$	Mem
$P \equiv_b P'$ and $P' \rightarrow_b Q'$ and $Q' \equiv_b Q$ implies $P \rightarrow_b Q$	Struct

In what follows we explain the rules of Table 3. The action $pino(\rho)$ creates an empty bubble within the membrane where the *pino* action resides. The original membrane buckles towards inside and pinches off; the patch σ on the empty bubble is a parameter of *pino*. The exo action $n^{\nwarrow}$ comes with a complementary co-action $\overline{n}^{\nwarrow}$; they model the merging of two nested membranes which starts with the membranes touching at a point. In this process (which is a smooth and continuous process), the subsystem P gets expelled to the outside, and all the residual patches of the two membranes become contiguous. The phago action $n^{\searrow}$ comes with a complementary co-action $\overline{n}^{\searrow}(\rho)$; they model a membrane (the one with Q) "eating" another membrane (the one with P). Again, the process has to be smooth and continuous, i.e., biologically implementable. It proceeds by the Q membrane wrapping around the P membrane and joining itself on the other side.Thus an additional layer of membrane is created around the eaten membrane: the patch on that membrane is specified by the parameter ρ of the co-phago action (similar to the parameter of pino action).

2 Mobility in Membrane Computing

Membrane systems are inspired by the living cells compartments. Membrane systems contain multisets of *objects*, *evolution rules* and possibly other membranes [17]. The model is inspired by biology, and uses rules as in formal languages theory. This is only one aspect that makes the model different from process calculi. Other differences are given by the parallel application of rules, depending also on the available "resources". The computations are performed in the following way: starting from an initial structure, the system evolves by applying the rules in a nondeterministic and maximally parallel manner; a rule is applicable when all the involved objects and membranes appearing in its left hand side are available. The maximally parallel way of using the rules means that in each step we apply a maximal multiset of rules, namely a multiset of rules such that no further rule can be added to the set. A halting configuration is reached when no rule is applicable, and he result is represented by the number of objects associated to a specified membrane. A specific feature of this model is that we can prove computability results in terms of Turing machines rather by reduction to the λ-calculus (as in the case of process calculi with mobility).

Mobile membranes represent a rule-based parallel computing model inspired by cells and their movements in which mobility is given by specific endocytosis and exocytosis rules [13]. Several systems of mobile membranes are studied in [3], and their computational universality are proved by using a small number of membranes [4].

Mobile membranes are characterized by two essential features:

- A spatial structure consisting of a hierarchy of membranes (which do not intersect) with objects associated to them; a membrane without any other membranes inside is called elementary.
- The general rules describing the evolution of the structure, namely endocytosis (moving an elementary membrane inside a neighbouring membrane) and exocytosis (moving an elementary membrane outside the membrane where it is placed). More specific rules can be defined.

In terms of computation, we are working with membrane configurations. We define the set $\mathcal{M}$ of membrane configurations (ranged by $M, N, \ldots$) by using the free monoid V^* (ranged over by $u, v, \ldots$) generated by a finite alphabet V (ranged over by $a, b, \ldots$): $M ::= u \mid [\, M \,]_u \mid M \| M$

If M and N are two membrane configurations, we say that M reduces to N (denoted by $M \rightarrow N$) if there exists a rule in the set of rules R applicable to the configuration M such that we can obtain the new configuration N. When applying the rules of R, we use also the following inference rules:

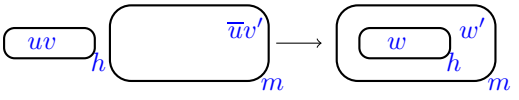
$$\textit{(Comp1)} \quad \frac{M \rightarrow M'}{M \| N \rightarrow M' \| N}; \quad \textit{(Comp2)} \quad \frac{M \rightarrow M' \qquad N \rightarrow N'}{M \| N \rightarrow M' \| N'};$$

$$\textit{(Mem)} \quad \frac{M \rightarrow M'}{[\, M \,]_u \rightarrow [\, M' \,]_u}; \quad \textit{(Struc)} \quad \frac{M \equiv_{mem} M' \quad M' \rightarrow N' \quad N' \equiv_{mem} N}{M \rightarrow N}$$

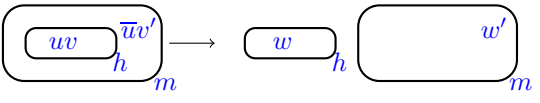
where the structural congruence $\equiv_{mem}$ over $\mathcal{M}$ is the smallest congruence relation such that $(\mathcal{M}, \|, \lambda)$ is a commutative monoid. When describing a computation given by a systems of mobile membranes, we start from an initial configuration M_0 and a set of rules.

We shortly present here a variant of systems with mobile membranes defined by *mutual mobile membranes*. These membranes use the following mobility rules:

Mutual Endocytosis: $[uv]_h[\overline{u}v']_m \rightarrow [\,[w]_h w']_m$



Mutual Exocytosis: $[[uv]_h\overline{u}v']_m \rightarrow [w]_h[w']_m$



Mutual mobile membranes can encode safe ambients. A translation $\mathcal{T}$ from the set $\mathcal{SA}$ of safe ambients to the set $\mathcal{M}^3$ of membrane configurations is given formally by $\mathcal{T}(A) = dlock\ \mathcal{T}_1(A)$, where $\mathcal{T}_1 : \mathcal{SA} \rightarrow \mathcal{M}^3$ is

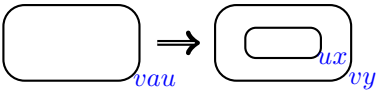
$$\mathcal{T}_1(A) = \begin{cases} cap\ n[\ ]_{cap\ n} & \text{if } A = cap\ n \\ cap\ n[\ \mathcal{T}_1(A_1)\ ]_{cap\ n} & \text{if } A = cap\ n.\ A_1 \\ [\ \mathcal{T}_1(A_1)\ ]_n & \text{if } A = n[\ A_1\ ] \\ [\ ]_n & \text{if } A = n[\ ] \\ \mathcal{T}_1(A_1), \mathcal{T}_1(A_2) & \text{if } A = A_1 \mid A_2 \end{cases}$$

An object *dlock* is placed near the membrane structure to prevent the consumption of capability objects in a membrane system which corresponds to a mobile ambient which cannot evolve further. Several results are presented in [2]. We mention only the operational correspondence.

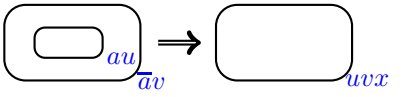
Theorem 1. *1. If $A \rightsquigarrow B$, then $\mathcal{T}(A) \rightarrow \mathcal{T}(B)$.*
2. If $\mathcal{T}(A) \rightarrow M$, then exists B such that $A \rightsquigarrow B$ and $M = \mathcal{T}(B)$.

Another variant of systems with mobile membranes is given by *mutual mobile membranes with objects on surface*. They use the following mobility rules:

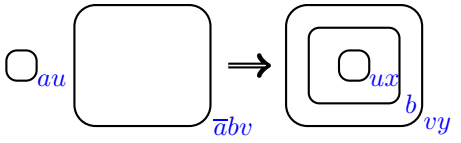
Pino: $[\ ]_{vau} \Rightarrow [[\ ]_{ux}]_{vy}$



Exo: $[[\ ]_{au}]_{\overline{a}v} \Rightarrow [\ ]_{uvx}$



Phago: $[\ ]_{au}[\ ]_{\overline{a}bv} \Rightarrow [[[\ ]_{ux}]_b]_{vy}$



The number of objects from the right hand side of a rule is called its *weight*. There are several results regarding the computational power of mutual membranes with objects on surface. We investigate the computational power of systems of mutual membranes with objects on surface controlled by pairs of rules: (pino,exo) and (phago,exo), proving that they are universal even using a small number of membranes. These cases were investigated initially in [12]; then we have obtained better results by improving the number of membranes. A summary of the results can be given by the following general sentence and the associated table.

Theorem 2. *Mutual membranes with objects on surface using n membranes and pair of operations (op_1, op_2) of weights (w_1, w_2) have the same computational power as a Turing Machine.*

Number n of membranes	Operations (op_1, op_2)	Weights (w_1, w_2)	Article
8	Pino, exo	4,3	Theorem 6.1 [Krishna07]
3	Pino, exo	5,4	Theorem 1 [AmanCiobanu09]
9	Phago, exo	5,2	Theorem 6.2 [Krishna07]
9	Phago, exo	4,3	Theorem 6.2 [Krishna07]
4	Phago, exo	6,3	Theorem 2 [AmanCiobanu09]

Mutual membranes with objects on surface can encode the processes of the PEP fragment of brane calculus without replication. A translation $\mathcal{T}$ from the set $\mathcal{PEP}$ of brane processes to the set $\mathcal{M}^2\mathcal{OS}$ of membrane configurations is given by:

$$\mathcal{T}(P) = \begin{cases} [\mathcal{T}(P)]_{\mathcal{S}(\sigma)} & \text{if } \sigma(P) \\ \mathcal{T}(Q)\ \mathcal{T}(R) & \text{if } P = Q \mid R \end{cases}$$

where $\mathcal{S} : \mathcal{PEP} \to \mathcal{M}^2\mathcal{OS}$ is defined as:

$$\mathcal{S}(\sigma) = \begin{cases} \sigma & \text{if } \sigma = n^{\searrow} \text{ or } \sigma = n^{\nwarrow} \text{ or } \sigma = \overline{n}^{\nwarrow} \\ \overline{n}^{\searrow}\ \mathcal{S}(\rho) & \text{if } \sigma = \overline{n}^{\searrow}(\rho) \\ pino\ \mathcal{S}(\rho) & \text{if } \sigma = pino(\rho) \\ \mathcal{S}(a)\ \mathcal{S}(\rho) & \text{if } \sigma = a.\rho \\ \mathcal{S}(\tau)\ \mathcal{S}(\rho) & \text{if } \sigma = \tau \mid \rho \end{cases}$$

The rules of the systems of mutual membranes with objects on surface are

$$[\,]_{S(n\searrow\sigma|\sigma_0)}[\,]_{S(\overline{n}\searrow(\rho).\tau|\tau_0)} \to [[[\,]_{S(\sigma|\sigma_0)}]_{S(\rho)}]_{S(\tau|\tau_0)}$$
$$[[\,]_{S(n\searrow.\sigma|\sigma_0)}]_{S(\overline{n}\searrow.\tau|\tau_0)} \to [\,]_{S(\sigma|\sigma_0|\tau|\tau_0)}$$
$$[\,]_{S(pino(\rho).\sigma|\sigma_0)} \to [[\,]_{S(\rho)}]_{S(\sigma|\sigma_0)}\ .$$

Proposition 1. *1. If $P \equiv_b Q$ then $\mathcal{T}(P) \equiv_m \mathcal{T}(Q)$, where $\equiv_m$ is the structural congruence over $\mathcal{M}^2\mathcal{OS}$.*
2. If $\mathcal{T}(P) \equiv_m M$ then there exists Q such that $M = \mathcal{T}(Q)$.

Moreover, we have an operational correspondence between these models.

Theorem 3. *1. If $P \Rightarrow Q$ then $\mathcal{T}(P) \to \mathcal{T}(Q)$.*
2. If $\mathcal{T}(P) \to M$ then there exists Q such that $P \to_b Q$ and $M \equiv_m \mathcal{T}(Q)$.

As a consequence of the fact that we translate a formalism using an interleaving semantic into a formalism working in parallel, it is possible to have Q such that $M = \mathcal{T}(Q)$, but $P \not\to_b Q$. Let us assume $P = \overline{n}\nwarrow.\overline{n}\nwarrow(n\nwarrow.n\searrow(\,))$. By translation, we obtain $M = ((\,)_{n\nwarrow \parallel n\searrow})_{\overline{n}\nwarrow \parallel \overline{n}\nwarrow}$, such that $M \to [\,]_{n\searrow \parallel \overline{n}\nwarrow} = N$. We observe that there exist $Q = n\searrow.\overline{n}\nwarrow(\,)$ such that $N = \mathcal{T}(Q)$, but $P \not\to_b Q$.

3 Conclusion

Several computational models have various notions of mobility in computer science. Here we have mentioned the π-calculus and distributed π-calculus, ambient calculus, brane calculi and membrane systems. The π-calculus, mobile ambients and brane calculi are closer to the λ-calculus. Membrane systems are strongly influenced by grammar rules, providing an automata and formal language view. All of them are essentially models of distributed, parallel and nondeterministic systems. Some of them have been inspired from the structure and the functioning of the living cell. We have provided translations of safe mobile ambients into mutual mobile membranes, and of PEP fragment of brane calculus into mutual membranes with objects on surface. Thus we prove that mobile membranes has at least the same expressive power as safe ambient calculus and PEP fragment of brane calculus. It seems that there is no significant expressiveness difference between these models; however there are also few differences; e.g., ambient and brane calculi have an interleaving semantic, while membrane systems have a parallel one.

Finally we can say that mobile membranes represent a computing model aiming to put together the advantages of both P systems and process calculi with mobility such as mobile ambients and brane calculi. Computations with mobile membranes are defined over specific configurations (like in process calculi), and represent a rule-based formalism (like P systems).

Acknowledgements

Several results regarding the P systems with mobile membranes were obtained during the last years together with Bogdan Aman; I express my gratitude to Bogdan for his work. Many thanks to all my process calculi collaborators.

This research work was supported by the CNCSIS project IDEI 402/2007.

References

1. Aman, B., Ciobanu, G.: Timed Mobile Ambients for Network Protocols. In: Suzuki, K., Higashino, T., Yasumoto, K., El-Fakih, K. (eds.) FORTE 2008. LNCS, vol. 5048, pp. 234–250. Springer, Heidelberg (2008)
2. Aman, B., Ciobanu, G.: On the Relationship Between Membranes and Ambients. Biosystems 91, 515–530 (2008)
3. Aman, B., Ciobanu, G.: Simple, Enhanced and Mutual Mobile Membranes. In: Priami, C., Back, R.-J., Petre, I. (eds.) Transactions on Computational Systems Biology XI. LNCS(LNBI), vol. 5750, pp. 26–44. Springer, Heidelberg (2009)
4. Aman, B., Ciobanu, G.: Turing Completeness Using Three Mobile Membranes. In: Calude, C.S., Costa, J.F., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) UC 2009. LNCS, vol. 5715, pp. 42–55. Springer, Heidelberg (2009)
5. Cardelli, L.: Brane calculi. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–278. Springer, Heidelberg (2005)
6. Cardelli, L., Gordon, A.: Mobile Ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)
7. Ciobanu, G., Krishna, S.N.: Enhanced Mobile Membranes: Computability Results. Theory of Computing Systems (to appear, 2011)
8. Ciobanu, G., Prisacariu, C.: Timers for Distributed Systems. Electronic Notes in Theoretical Computer Science 164, 81–99 (2006)
9. Ciobanu, G., Rotaru, M.: Molecular Interaction. Theoretical Computer Science 289, 801–827 (2002)
10. Ciobanu, G., Zakharov, V.: Encoding Mobile Ambients into π-calculus. In: Virbitskaite, I., Voronkov, A. (eds.) PSI 2006. LNCS, vol. 4378, pp. 148–161. Springer, Heidelberg (2007)
11. Hennessy, M.: A Distributed π-calculus. Cambridge University Press, Cambridge (2007)
12. Krishna, S.N.: Universality Results for P Systems Based on Brane Calculi Operations. Theoretical Computer Science 371, 83–105 (2007)
13. Krishna, S.N., Păun, G.: P Systems with Mobile Membranes. Natural Computing 4, 255–274 (2005)
14. Levi, F., Sangiorgi, D.: Mobile Safe Ambients. ACM TOPLAS 25, 1–69 (2003)
15. Milner, R.: Elements of Interaction. Turing Award Lecture. ACM Comm. 36, 78–89 (1993)
16. Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press, Cambridge (1999)
17. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
18. Păun, G., Rozenberg, A., Salomaa, A.: Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
19. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: BioAmbients: An Abstraction for Biological Compartments. Theoretical Computer Science 325, 141–167 (2004)
20. Regev, A., Shapiro, E.: The π-calculus as an Abstraction for Biomolecular Systems. In: Ciobanu, G., Rozenberg, G. (eds.) Modelling in Molecular Biology. Natural Computing Series. Springer, Heidelberg (2004)
21. Victor, B., Moller, F.: The Mobility Workbench – A Tool for the π-calculus. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 428–440. Springer, Heidelberg (1994)

Organization Oriented Chemical Computing

Peter Dittrich

Friedrich-Schiller University Jena
Institute of Computer Science
Bio Systems Analysis Group
Ernst-Abbe-Platz 14, 07743 Jena, Germany
peter.dittrich@uni-jena.de

All known life forms process information on a bio-molecular level, which is known to be robust, self-organizing, adaptive, decentralized, asynchronous, fault tolerant, and evolvable. In order to exploit these properties it has been suggested to use artificial chemical systems like P-systems or artificial hormone systems. However, finding the right chemical program appears to be difficult, because computation emerges out of an interplay of many decentralized relatively simple components called molecules. Therefore, a human programmer needs paradigms and tools that allow to predict the behavior of a chemical program, i.e., a set of reaction and transformation rules. In this talk, I focus on how chemical organization theory (*Bull. Math. Biol.*, 69, 1199-, 2007) can help in designing and understanding chemical computing systems. The theory decomposes reaction networks into a hierarchy of closed and self-maintaining sub-networks called organizations. For this analysis stoichiometric information is sufficient, which can be directly obtained from the reaction rules and which is usually independent from kinetics, e.g., a systems state. The fundamental idea of organization oriented chemical programming is to view chemical computing as a movement between organizations (*Int. J. Unconv. Comp.*, 3(4), 285-309, 2007). Our hypothesis is, if the behavior of a chemical program can be explained in terms of organizations, it will be dynamically robust. This hypothesis is supported by a theorem stating that all long-term behavior of a chemical ordinary differential equation will end up in an organization. I will discuss the approach using examples like a chemical XOR, a flip-flop, a controllable oscillator, and the maximum independent set problem, which could be relevant in a distributed sensor networks. Finally, as an open problem, I will discuss how it might be possible to derive chemical organizations directly from implicit reaction rules.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, p. 18, 2010.

Cellular Automata and the Quest for Nontrivial Artificial Self-Reproduction

Markus Holzer and Martin Kutrib

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{holzer,kutrib}@informatik.uni-giessen.de

Abstract. The quest for artificial self-reproduction dates back to the end of the 1940's and started with the work of John von Neumann on self-reproducing cellular automata. Nowadays (artificial) self-reproduction is one of the cornerstones of automata theory, which plays an important role in the field of molecular nanotechnology. We briefly summarize the development on the research subject of artificial self-reproduction starting with von Neumann's ideas. Moreover, we pay special attention to the concepts of trivial and non-trivial self-reproduction by Herman, Langton, and others. Our tour on the subject obviously lacks completeness and it reflects our personal view of what constitute the most interesting links to the important aspects of artificial self-reproduction.

1 Introduction

More than half a century ago John von Neumann had become interested in the question of whether computing machines can construct copies or variants of themselves, that is, whether artificial self-reproducing structures exist. He started to investigate what kind of logical organization is sufficient for an automaton to control itself in such a manner that it reproduces itself, and found one of the cornerstones in the theory of automata. It is remarkable that this problem was posed at a time when even the concept of a universal programmable computer was in its infancy. At that time DNA had not yet been discovered as the genetic material in nature, nor did von Neumann have the tools for building a real machine at the bio-chemical or genetic level. However, the following five fundamental questions raised by him [18] reveal the vision he had in mind. Nowadays we have to see the questions with the eyes of von Neumann, in particular, with the eyes of a man who lived 60 year ago.

Fundamental Questions [John von Neumann 1949]

Logical universality. When is a class of automata logically universal, i.e., able to perform all those logical operations that are at all performable with finite (but arbitrarily extensive) means? Also, with what additional—variable, but in the essential respects standard—attachments is a single automaton logically universal?

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 19–36, 2010.

Constructibility. Can an automaton be constructed, i.e., assembled and built from appropriately defined "raw materials," by another automaton? Or, starting from the other end and extending the question, what class of automata can be constructed by one, suitably given, automaton? The variable, but essentially standard, attachments to the latter, in the sense of the second question of *Logical universality*, may here be permitted.

Construction-universality. Making the second question of *Constructibility* more specific, can any one, suitably given, automaton be construction-universal, i.e., be able to construct in the sense of question *Constructibility* (with suitable, but essentially standard, attachments) every other automaton?

Self-reproduction. Narrowing question *Self-reproduction*, can any automaton construct other automata that are exactly like it? Can it be made, in addition, to perform further tasks, e.g., also construct certain other, prescribed automata?

Evolution. Combining questions *Construction-universality* and *Self-reproduction*, can the construction of automata by automata progress from simpler types to increasingly more complicated types? Also, assuming some suitable definition of "efficiency," can this evolution go from less efficient to more efficient automata?

While the answer to the first question was known from Turing's famous paper [16,17], von Neumann established affirmative answers to the questions *Constructibility*, *Construction-universality*, and *Self-reproduction*. Basically, he considered five models of self-reproduction, the kinematic, cellular, excitation-threshold-fatigue, and continuous model [18], but mainly dealt with the kinematic and the cellular model only.

Von Neumann's Kinematic Model

In the kinematic model established as a thought experiment *kinematic automata* are composed of different types of primitive elements [5]: switches as *and*, *or*, *not*, and *delay* gates are used to form primitive computing elements, muscle-like elements (for example, an artificial hand), which can move elements around when signaled to do so by a computing element, cutting elements that can disconnect two elements, fusing elements which can connect two (fitting) elements together, rigid elements that provide structural support to assemblies of elements, and sensing elements capable of recognizing each kind of element and communicating this information to a computing element.

In order to design a self-reproducing kinematic automaton an environment is necessary with which the automaton can interact. Assume the environment to be an infinite hardware soup composed of the same ingredients as the automaton, that is, the various primitive elements. So an indefinite supply of parts is available to any kinematic automaton floating in the soup. Such an environment together with the floating kinematic automata is called a *kinematic system*.

Fig. 1. Kinematic tapes representing 0110110 (left) and 1101001 (right)

Any finite kinematic automaton can be described by listing its part and its connections. The binary blueprint can be stored on a *kinematic tape*, that is, a zig-zag sequence of rigid elements, where at each intersection there is a protruding rigid element if and only if a 1 is represented (see Figure 1). Now, a kinematic automaton can move itself relative to the tape by means of kinematic elements, it can change a 1 to a 0 by cutting the rigid element from the tape, it can change a 0 to a 1, or extend the tape, by sensing a rigid element from the hardware soup with a sensing element, picking it up and placing it in position with a muscle-like element, and connecting it to the tape by a fusing element. Therefore, suppose a constructing automaton that reads the blueprint on the kinematic tape, interprets it by its computing elements, and assembles an automaton according to the specification of the blueprint. Since the constructing automaton is itself a kinematic automaton, it can construct a copy of itself if its own description is stored on the tape. The whole system consisting of the automaton and the tape containing the blueprint is self-reproducing if, after the construction of the new automaton, the constructing automaton makes a tape for the new machine, copies its own tape content to the new tape and attaches it to the new machine.

The main problem with this kinematic model is its impreciseness. It lacks the possibility to be treated mathematically in detail. In order to work out a detailed construction in a logically rigorous way, the powers of each element and the rules of its operation, the interactions with the environment, etc., must completely be specified. This appeared to be a much to complex task to deal with mathematically. So, von Neumann gave it up. However, since it is close to our intuitive understanding of artificial self-reproduction, let us look at its core.

Basically, the raw material in the environmental hardware soup should be as elementary as possible. Into this environment the artificial system is introduced, which itself is composed of this raw material. The system then organizes another area of the environment according to its instructions. At the end of the process the new area is a replica of the system. Having this in mind a step to a more abstract model can be done. Suggested by Ulam, von Neumann changed to the *cellular model*, which is less vivid and dramatic, less realistic, and less difficult to deal with mathematically [11]. However, it eliminates the complexities of the kinematic model. He employed a mathematical device which is a multitude of interconnected finite-state machines operating in parallel to form a larger machine. He showed that it is logically possible for such a nontrivial computing device to replicate itself ad infinitum. This was the birth of cellular automata.

2 Cellular Automata

Basically, the idea of cellular automata is to consider a universe which is a two-dimensional infinite array of deterministic finite automata, the *cells*. In order to keep the system tractable, a high degree of homogeneity is preferable. Therefore, all cells are assumed to be identical, and a unique interconnection scheme defines the cells which are connected to any given cell. Eventually, the cells operate synchronously at discrete time steps obeying a local transition function, which maps the current state of the cell itself and the current states of its connected cells (neighbors) to the next state. There exists a special, so-called *quiescent state* with the property that if some cell and all of its neighbors are in the quiescent state then the cell remains in the quiescent state.

Before we turn to define cellular automata formally, we clarify the notions of automata, machines, and environment in cellular automata. The single cells are the raw material from which larger machines have to be built. So, the infinite array is the environment, the hardware soup. Assume for a moment that all cells of the array are in the quiescent state, that is, the computation of the cellular automaton as a whole is stable. Now a larger machine is introduced in the environment by adjusting the states of the cells in an area. These cells form a machine which is clearly built from raw material. It may interact with the environment by letting initially quiescent cells in the neighborhood change their states according to the given local transition function. So, a multitude of finite automata operating in parallel form a larger machine such that the global behavior is achieved by local interactions only. In the cellular model a finite connected area of cells corresponds to a kinematic automaton in the kinematic model. Similarly, the whole array including the embedded machine corresponds to a kinematic system.

In order to be general, we define cellular automata formally over arbitrary dimensions. So, assume that the cells of a cellular space are arranged as a d-dimensional grid such that we deal with the Euclidean space $\mathbb{Z}^d$. In this way cells are identified by their integer coordinates.

Definition 1. *A* d-dimensional cellular automaton *is a system* $\langle S, d, N, \delta, q_0 \rangle$, *where*

1. S *is the finite, nonempty set of* cell states,
2. $d \in \mathbb{N}$ *is the* dimension,
3. $N = (n_1, n_2, \ldots, n_k)$, *with different components* $n_1, \ldots, n_k \in \mathbb{Z}^d$, *is the* d-dimensional neighborhood-index of degree k.
4. $\delta : S^k \to S$ *is the* local transition function, *and*
5. $q_0 \in S$ *is the* quiescent state *such that* $\delta(q_0, q_0 \ldots, q_0) = q_0$.

In general, the global behavior of a cellular automaton is of interest. It is induced by the local behavior of all cells, that is, by the local transition function. More precisely, a *configuration* of a cellular automaton $\langle S, d, N, \delta, q_0 \rangle$ is a description of its global state, which is formally a mapping $c : \mathbb{Z}^d \to S$. Successor

configurations are computed according to the global transition function Δ. Let c be a configuration. Then its successor $c' = \Delta(c)$ is defined by

$$c'(i) = \delta(c(i + n_1), \ldots, c(i + n_k)), \text{ for all } i \in \mathbb{Z}^d.$$

So, in order to identify the neighbors of a cell i one has to add the elements of N to i. In particular, if 0 belongs to N, then cell i is its own neighbor. Only in this case the next state of a cell depends on its current state. Figure 2 shows two generalized types of neighborhood-indices. The *von-Neumann neighborhood* has been used by von Neumann [18] and the Moore-neighborhood has been introduced by Moore in [11].

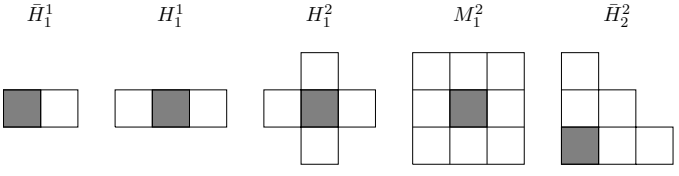


Fig. 2. Generalized von-Neumann (H_r^d) and Moore (M_r^d) neighborhoods. The origin is shaded, d is the dimension, and r denotes the 'radius'. A bar indicates a restricted version where all coordinates of the components are non-negative. For example, $\bar{H}_1^1 = (0, 1)$, $H_1^1 = (-1, 0, 1)$, $H_1^2 = ((0, 0), (0, 1), (1, 0), (0, -1), (-1, 0))$, $H_1^1 = (-1, 0, 1)$, and $M_1^2 = ((0, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1), (-1, 0))$.

The following example of a cellular automaton is know as the famous *Game of Life* (see, for example, [4,7]). While the underlying rules are quite simple, the global behavior is rather complex. In fact, it is unpredictable.

Example 1. We consider a two-dimensional automaton, where the cells are connected according to the Moore-neighborhood M_1^2, that is, each cell is connected to itself and to its eight immediate neighbors. The state set is $\{0, 1\}$. The local transition function is defined by the sum of the states of the neighbors and of the cell itself. In particular, a cell enters state 1, if the sum is three. it keeps its current state, if the sum is four, and enters state 0 in all other cases. Figure 3 shows the evolution of a pattern (computation of an embedded machine) such

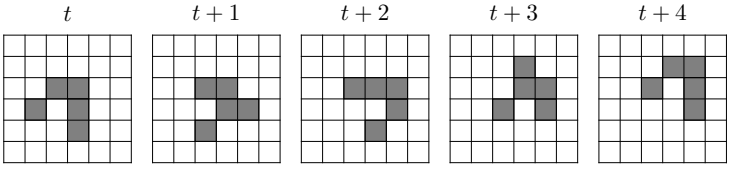


Fig. 3. An embedded machine working according to the local transition function of the Game of Life moving to the northeast. Empty cells are in state 0, shaded cells are in state 1.

that its position moves one cell to the right and one cell to the left every fourth computation step. Embedded in an empty, that is, quiescent environment, the machine moves across the environment forever. □

3 Von Neumann's Universal Constructor

Next, we formalize nontrivial self-reproduction and based on these definitions we then present the basic properties of von Neumann's cellular automaton.

3.1 The Notion of Nontrivial Self-Reproduction

Let c be a configuration. Then the set of all non-quiescent cells of c is said to be the *support* of c, which is denoted by $\sup(c)$. A configuration c' is a *subconfiguration* of c if $c|_{\sup(c')} = c'|_{\sup(c')}$. Configuration c is called *passive* if $\Delta(c) = c$, and *completely passive* if every subconfiguration of c is passive. Two configurations c and c' are said to be *disjoint* if $\sup c \cap \sup c' = \emptyset$. The *union* of two disjoint configurations c and c' is defined by

$$(c \cup c')(i) = \begin{cases} c(x) & \text{if } x \in \sup(c) \\ c'(x) & \text{if } x \in \sup(c') \\ q_0 & \text{otherwise,} \end{cases}$$

where q_0 is the quiescent state.

Definition 2. *Let c and c' be two configurations. Configuration c* constructs *c' if there is a time t such that c' is a subconfiguration of $\Delta^t(c)$ disjoint to c.*

The requirement that c' has to be disjoint from c eliminates the trivial case that completely passive configurations constructing itself at every time step, and also the case in which non-quiescent cells become quiescent. In [6] a stronger definition is given. There, in addition, it is required that after a certain time $\Delta^t(c)$ has no longer effect on c'.

Definition 3. *A configuration c is* self-reproducing *if c constructs c.*

The situation so far is somehow unsatisfactory as the following example of a trivial self-reproduction reveals.

Example 2 (Trivial self-reproduction). Let $\langle \{q_0, 1, 2\}, 2, N, \delta, q_0 \rangle$ be a cellular automaton with $N = H_1^2 = ((0,0),(0,1),(1,0),(0,-1),(-1,0))$ and

$$\begin{aligned} &\delta(q_0, 1, s_2, q_0, s_4) = 2, \\ &\delta(q_0, 1, s_2, 1, s_4) \ = q_0, \\ &\delta(q_0, 2, s_2, s_3, s_4) = 1, \\ &\delta(1, s_1, s_2, s_3, s_4) = 1, \text{ and} \\ &\delta(2, s_1, s_2, s_3, s_4) = q_0, \text{ for } s_1, s_2, s_3, s_4 \in \{q_0, 1, 2\} \end{aligned} \quad .$$

Basically, a quiescent cell enters the intermediate state 2 when its upper neighbor is in state 1. Subsequently it becomes quiescent again and its lower neighbor enters state 1. A quiescent cell with upper and lower neighbor both in state 1remains quiescent, and a cell in state 1 always remains in state 1. So, for any fixed $j_0 \in \mathbb{Z}$ all configurations of the form $c((i,j)) = q_0$ for $i \in \mathbb{Z}, j \neq j_0$, and $c((i,j_0)) \in \{q_0, 1\}$ for all $i \in \mathbb{Z}$, are trivially self-reproducing. That is, a line of the array constructs a disjoint copy of itself two cells downward. □

In order to cope with the problem of trivial self-reproduction one can require that the self-reproducing system is able to perform other tasks, for example, to compute recursive functions or to construct other machines according to a blueprint. This in turn yields the necessity to deal with problem how to represent data such as arguments of functions or blueprints of machines. It is reasonable to represent data as completely passive configurations in order to ensure that the data cannot modify itself in any way. So, a *Turing domain* T for a cellular automaton is an infinite set of completely passive configurations with an effective computable bijection from $\mathbb{N}$ to T. It follows that T permits to encode any information that could be required. Let $\sup(T)$ be $\bigcup_{d \in T} \sup(d)$.

Definition 4. *A configuration c is a* universal constructor *for a class C of configurations if for every $c' \in C$ there exists a $d \in T$ such that $c \cup d$ constructs c'.*

In order to complete the clarification of notions we next consider the computation of functions in cellular automata.

Definition 5. *Given a cellular automaton A with Turing domain T, a partial function ψ from T into T is* computable *in A, if there is a configuration c_ψ, a so-called ψ-*computer*, disjoint from all elements in T, such that for every $d \in T$, $\psi(d)$ is defined if and only if there is a $t \in \mathbb{N}$ such that $\Delta^t(c_\psi \cup d)|_{\sup(T)}$ is passive and belongs to T.*

Figure 4 illustrates the computation of a function in a cellular automaton.

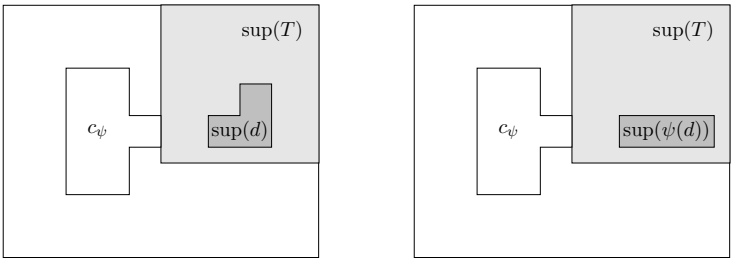


Fig. 4. Computation of a function ψ in a cellular automaton. The configuration at the left evolves in some t steps into the configuration at the right.

Definition 6. *A cellular automaton A is* computation universal, *if there is a Turing domain T for A such that every partial recursive function from T into T is computable in A. A configuration c is said to be a* universal computer *(with respect to T), if for any partial recursive function ψ from T into T there is a $d_\psi \in T$ (ψ-program), such that $c \cup d'_\psi$ is a ψ-computer, where d'_ψ is a copy of d_ψ disjoint from c and all elements from T.*

Figure 5 illustrates the principle of a universal computer in a cellular automaton.

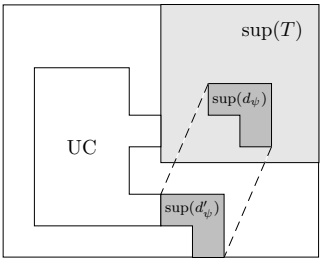


Fig. 5. Universal computer (UC) in a cellular automaton

In order to rule out trivial self-reproduction von Neumann required that the self-reproducing configuration also being a universal computer. In addition, each self-reproducing configuration should be a universal constructor for a large class of configurations. Clearly, this class has to include the self-reproducing configuration itself.

Analogous to the property of being computational universal one can ask for the property of being *construction universal*. Let A be a computation universal cellular automaton with Turing domain T, and C be the class of all ψ-computers in A. If any member of C is constructible by some configuration of A disjoint from all elements in C, then A is *construction universal*. Of particular interest is the case in which both a universal computer and a universal constructor exist and the set of tapes required by the universal constructor is included in T. A single configuration with these properties is called a *universal computer-constructor*.

3.2 Von Neumann's Cellular Automaton

John von Neumann succeeded to design a cellular automaton capable of non-trivial self-reproduction. His model uses 29 states and the five cell neighborhood-index H_1^2. The self-reproducing machine embedded in the cellular automaton is designed from several parts which he called "organs" involving a *central control unit*, a *tape unit* intended to read the tape containing the instructions, and a *constructing unit* which is basically a constructing arm to build the offspring (see Figure 6). The details can be found in the book [18] on more than 130

pages, partially in verbal and diagrammatic form. Note that a complete formal representation of the transition function would have 29^5, or approximately 20 million, entries. Due to their complexity the details are beyond the scope of this article.

However, roughly speaking, one basic module of the design principle are adjacent cells in certain transmission states that form data paths. These states can be passive or excited, that is, transmit an "electron" or not. For example, by different signal sequences the construction arm can be controlled such that it moves or writes at its end by changing the states of neighboring cells appropriately.

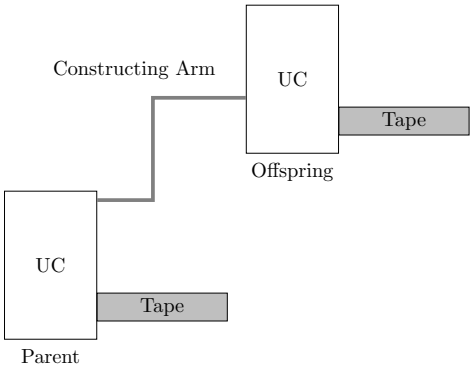


Fig. 6. Principle of von Neumann's self-reproducing automaton

The intuitive notion of self-reproduction was defined formally for the first time by Thatcher in [15]. Based on this definitions he verified the exciting properties of the von Neumann cellular automaton, which are presented in the following propositions.

Garden-of-Eden configurations are configurations that cannot appear as a result of an application of the global transition function. So, they can only appear at initial time. In [11] it is shown that the existence of a pair of *mutually erasable* configurations, that is, both have the same successor under the global transition function, implies the existence of a Garden-of-Eden configuration. Since there are Garden-of-Eden configurations in the von Neumann cellular automaton we obtain the following proposition.

Proposition 1. *There exist non-constructible configurations in the von Neumann cellular automaton.*

The process of constructing a configuration in the von Neumann cellular automaton is as follows. A description of the configuration to be constructed is stored in explicit form on the tape of the universal constructor. Under direction of the central control unit, the tape unit reads the description from the tape and transmits it to the constructing unit. Using this description, the constructing unit

sends signals into the constructing arm to bring out the construction. To this end, the configuration to be constructed has to be completely passive.

Proposition 2. *There exists a universal constructor for the class of completely passive configurations in the von Neumann cellular automaton.*

In order to obtain self-reproduction, first it has to be mentioned that the universal constructor itself is basically a completely passive configuration that has initially to be activated. So, it becomes a self-reproducing configuration by extending its functionality as follows. When it is started with its own description on the input tape, the description is executed and thereby a completely passive (inactive) copy of the universal constructor is constructed. Driven by a code on the tape the constructor will proceed to construct a copy of its own tape and attaches it to the already constructed configuration at some standard place. After the tape has been constructed the constructing arm will return to the constructed configuration and give it an activation signal.

Proposition 3. *There exists a self-reproducing universal constructor in the von Neumann cellular automaton.*

On passing, the design of the universal constructor shows how to embed an arbitrary Turing machine in the von Neumann cellular automaton.

Proposition 4. *The von Neumann cellular automaton is computation universal.*

In particular a universal Turing machine can be embedded.

Proposition 5. *There exists a universal computer in the von Neumann cellular automaton.*

Since a universal Turing machine can be combined with the universal constructor we obtain the next appealing strong result.

Theorem 1. *There exists a self-reproducing universal computer-constructor in the von Neumann cellular automaton.*

The result of von Neumann has been refined and simplified by Thatcher who presented formal definitions of the underlying notions and the construction on about 50 pages [15]. Codd reduced the size of the cells to eight states in a proof of 80 pages [6]. Arbib reduced the length of the proof to eight pages, but paid for it with extremely large cells having 2^{335} states [2]. Conversely, Banks has shown on about 20 pages the existence of nontrivial self-reproducing machines in a cellular automaton with only four states [3]. In all these cases the neighborhood-index is H_1^2 including five cells, and non-triviality means that there exists a universal computer-constructor. From a theorem of Codd follows that there is no two-dimensional two-state cellular automaton with neighborhood-index H_1^2 capable of non-trivial self-reproduction [6]. However, for the nine-cell neighborhood-index M_1^2 non-trivial self-reproduction is possible with two

states [19] in the famous *Game of Life* (see, for example, [4,7]), provided the definition of computability is slightly relaxed [19]. The later is necessary since completely passive configurations cannot be used to represent data when only two states are available. Therefore, the data is represented by certain periodic configurations.

4 Trivial versus Nontrivial Self-Reproduction

Recalling the trivial self-reproducing cellular automaton of Example 2 it is intuitively clear that such self-reproducing structures are not significant either from the biological point of view or in answer to the problem of machine self-reproduction. Due to the vague nature of the problem the question came up what nontrivial self-reproduction is. Therefore, attention has been paid on designing self-reproducing configurations in cellular automata that additionally can carry out complex tasks. In order to formalize "complex tasks" the notion of a universal computer-constructor has been introduced, which combines universality of computations and constructions. So, a universal computer-constructor is anything but trivial. However, the power of being a self-reproducing universal computer-constructor is no guarantee for being a nontrivial self-reproducing configuration. Herman presented the following construction of a universal computer-constructor which is trivial self-reproducing [9].

Theorem 2. *There exists a cellular automaton A with a Turing domain T and a configuration u such that $|\sup(u)| = 1$, u is self-reproducing, and u is a universal computer-constructor.*

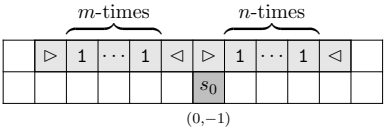
In order to design such a system Herman started with a universal Turing machine U with state set Q, tape alphabet P and blank symbol $\sqcup$ having the following properties: the initial state s_0 is never reached from any other state, there is a distinguished halting state s_f, that is, the transition function is not defined for s_f. Moreover, there are tape symbols 1, $\rhd$, and $\lhd$, and for all nonnegative integers m and n, if $\psi_m(n)$ is not defined then U starting in state s_0 scanning the right symbol $\rhd$ of the tape inscription $\rhd 1^m \lhd \rhd 1^n \lhd$ does not halt, and else U halts in state s_f scanning the right symbol $\rhd$ of the tape inscription $\rhd 1^m \lhd \rhd 1^{\psi_m(n)} \lhd$, and the initially scanned symbol $\rhd$ is never erased during the computation. Here ψ_m denotes the function mapping from the nonnegative integers to the nonnegative integers that is computed by the Turing machine with Gödel number m.

The cellular automaton $A = \langle S, 2, M_1^2, \delta, q_0 \rangle$ is defined by $S = Q \cup P \cup \bar{Q} \cup \hat{Q}$, where $\bar{Q} = \{ \bar{s} \mid s \in Q \}$ is a disjoint copy of Q, $\hat{Q}$ is defined later, and $q_0 = \sqcup$. The state transition δ is defined as we go along. With $M_1^2 = ((0,0),(-1,1),(0,1),(1,1),(1,0),(1,-1),(0,-1),(-1,-1),(-1,0))$ we set $\delta(s_0, \sqcup, x, \sqcup, \sqcup, \sqcup, y, \sqcup, \sqcup) = s_0$, for $x, y \in \{\sqcup, s_0\}$, and $\delta(\sqcup, s_0, \sqcup, \sqcup, \sqcup, \sqcup, \sqcup, \sqcup, \sqcup) = s_0$.

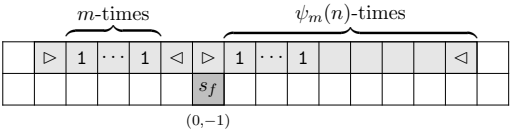
The configuration u is defined as $u(i) = s_0$ if $i = (0,-1)$, and $\sqcup$ otherwise. So, u is trivial self-reproducing.

In order to define the Turing domain T, let for any $n \geq 0$ a configuration d_n be as follows: $d_n(i) = \triangleright$ if $i = (0,0)$, $d_n(i) = 1$ if $i \in \{\,(0,j) \mid 1 \leq j \leq n\,\}$, $d_n(i) = \triangleleft$ if $i = (0, n+1)$, and $d_n(i) = \sqcup$ otherwise. Together with setting $\delta(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) = x_1$, for all $x_j \in P$ we obtain that T is in fact a Turing domain.

Now, the cellular automaton A is extended such that it simulates the Turing machine U step-by-step. To this end, it suffices to define δ for all remaining arguments of $Q \cup P$ in such a way that starting with the configuration



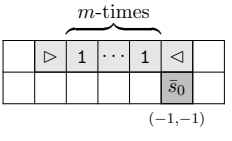
automaton A will reach the passive configuration



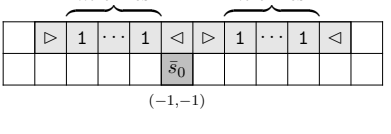
if $\psi_m(n)$ is defined, that is, if U halts, then automaton A runs into a passive configuration. Omitting details of the construction this shows that s_0 is a universal computer in A with Turing domain T.

Since the initial state s_0 is never reached from any other state by U, so far, δ is only defined for state s_0 in the case it has a $\triangleleft$ in its top left neighbor.

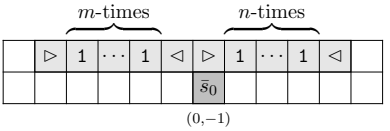
For any $m \geq 0$, let c_m denote the passive configuration



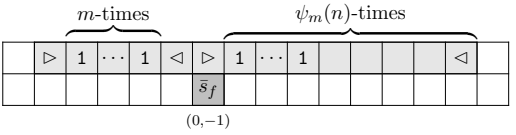
which is a ψ_m-computer when δ is chosen as follows. For any $d_n \in T$, starting with the configuration $c_m \cup d_n$ the cellular automaton A first moves the state $\bar{s}_0$ one cell to the right, and then simulates U by using states from $\bar{Q}$ instead of Q. If and only if the simulation halts, automaton A moves $\bar{s}_f$ one cell to the left. With other words, δ is defined for all arguments from $\bar{Q} \cup P$ in such a way that if $\psi_m(n)$ is defined then in its first step A transforms the configuration $c_m \cup d_n$



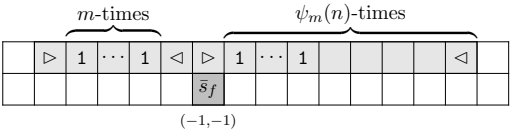
into



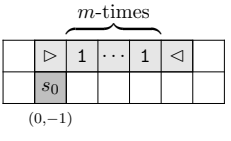
and subsequently in some t further time steps into



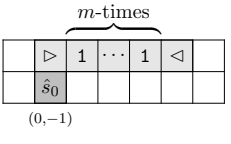
and finally into the passive configuration



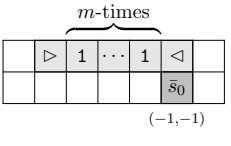
Now it remains to be shown that any ψ_m-computer of A is constructible by u. In particular, the set $\hat{Q}$ is used to construct, for any m, the configuration c_m by $u \cup d_m$. We define the so far undefined $\delta(s_0, \sqcup, \sqcup, \sqcup, x, \triangleright, \sqcup, \sqcup, \sqcup)$, for $x \in \{1, \triangleleft\}$, to be $\hat{s}_0$, and furthermore δ for all arguments from $\hat{Q} \cup P$ in such a way that the configuration $u \cup d_m$



in the first step of A is transformed to



and, after some t further steps into



Since this is the configuration c_m, it follows that for each computable partial function ψ_m from T into T, there exists a $d_m \in T$ such that $u \cup d_m$ constructs

the configuration c_m which, in turn, computes ψ_m. Therefore, u is also a universal constructor for the Turing domain T, and altogether u is a trivial self-reproducing universal computer-contractor as stated by the theorem.

So, what does the result show? Since without any doubt the von Neumann cellular automaton and its refined and simplified versions are nontrivial self-reproducing machines, the result shows that the existence of a self-reproducing universal computer-constructor in itself is not relevant to the problem of biological and machine self-reproduction.

We conclude this section with a question of Moore raised about half a century ago [11] for which a satisfiable answer is still open:

> What can be done to make the statement that one machine is more general or less trivial than another in its self-producing behavior more precise? There does not seem to be a clear-cut line of demarcation between the trivial and nontrivial models.

5 Self-Reproducing Loop Cellular Automata

Von Neumann was interested in the question as to what kind of logical organization is *sufficient* for an artificial machine to be able to reproduce itself. We have seen that the question is not precise and admits to trivial versions as well as interesting ones. So, the question as to what kind of logical organization is *necessary* for an artificial machine to be able to reproduce itself (in a nontrivial way) arises immediately. In the last section we have seen that one single cell can be a self-reproducing universal computer-constructor. But this one-cell machine is not simple, its logic includes the transition rules of a universal Turing machine.

Langton [10] argued that the approach to rule out trivial self-reproduction by requiring a universal computer-constructor also rules out all naturally occurring self-reproducing systems, since none of these have been shown to be capable of universal construction. He suggested another criterion which is satisfied by molecular self-reproduction: the configuration must treat its input information in two different manners, as *instructions to be interpreted* (translation) as well as *uninterpreted data* (transcription). This is what happens in the von Neumann cellular automaton. The instructions are executed in order to construct a machine, and they are copied and attached to the new machine. Langton proposed a cellular automaton using the five-cell H_1^2 neighborhood-index and a simple embedded configuration, which can effect it own reproduction by employing the process of transcription and translation [10].

The basic structure is a data-path similar as has been used by Codd [6]. A data-path consists of *core* cells surrounded by *sheath* cells as shown in Figure 7. Signals which consist of packages of two consecutive states (genes) travel along the path and are copied at junctions (see Figure 7).

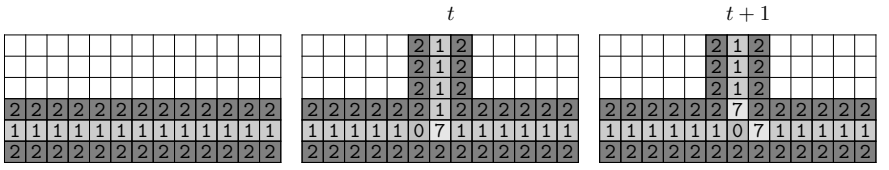


Fig. 7. On the left, a data-path is shown, where core cells are represented by state 1, and sheath cells by state 2. The two right drawings show that signals traveling along a data-path are copied at junctions.

The possible signals are 40, 50, 60, or 70, where 0 is the quiescent state. At the end of the path a signal is interpreted to extend the path. For example, when signal 70 arrives the path is extended by one cell, and two 40 signals make it turn left. In order to obtain self-reproduction, a loop with an exit (junction) is constructed in which signals cycle. Since they arrive at the junction at regular intervals, the problem to solve is to get a sequence of signals into the loop (that are copied once and again at the junction) which reproduce the loop. The solution of Langton is to consider a loop having the form of a square and to use the signal sequence 70,70,70,70,70,70,40,40. The six signals 70 extend the path by six cells, and the two 40 signals make it turn left. So, four copies of the sequence will construct the four sides of the new loop (see Figure 8).

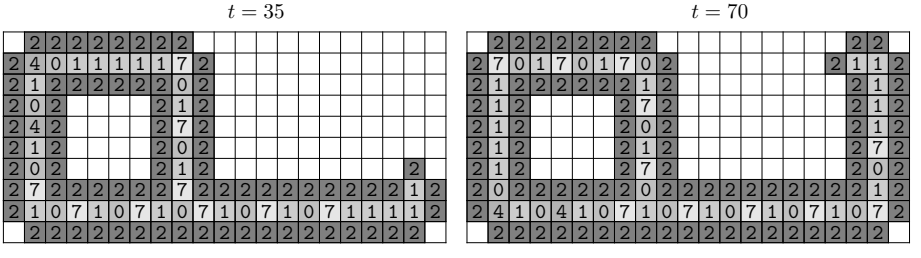


Fig. 8. Construction of an offspring loop

Finally, the offspring is separated by the collision of signals at the new junction. When this happens new signals are generated which separate the loops and initiate the construction of new exits in both parent and offspring loops (see Figure 9). If a loop tries to extend its arm to an occupied area, a sheath fragment is generated that absorbs all genes. The details of the construction including the transition function can be found in [10].

Under the criterion of nontriviality proposed by Langton the loop is capable of nontrivial self-reproduction. However, it is not capable to do anything else. So, even without knowing an answer to the question of Moore as quoted above one can certainly say that the loop is a trivial self-reproducing system.

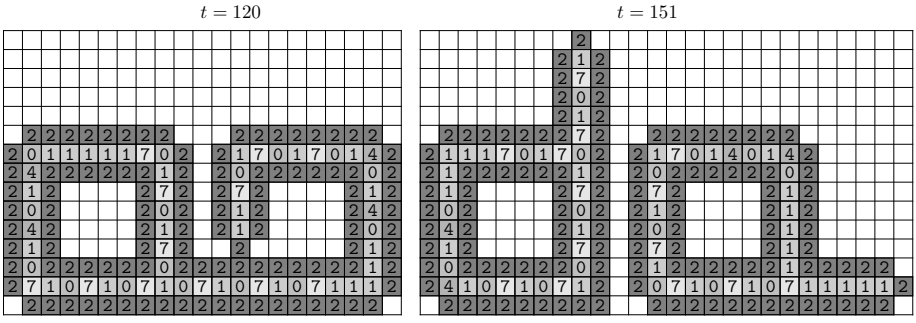


Fig. 9. Configurations after 120 and 151 steps. In the latter drawing the loop has reproduced itself exactly.

6 Reproduction of Arbitrary Configurations

This section is devoted to another aspect of self-reproduction in cellular automata. Are there cellular automata capable of reproducing *any* finite initial configuration? The question has been answered in the affirmative for one- and two-dimensional cellular automata by Amoroso and Cooper [1]. Generalizations to arbitrary dimensions have been shown in [8,13].

The main result of Amoroso and Cooper reads as follows.

Theorem 3. *For any set of states S containing at least two elements, and for dimension $d \in \{1, 2\}$, there exists a cellular automaton $\langle S, d, N, \delta, q_0\rangle$ that reproduces any configuration c with finite support. The reproduction takes place in a number t of steps that is a function of c, Further, if $|S|$ is prime the reproduced copies will be contained in a quiescent environment.*

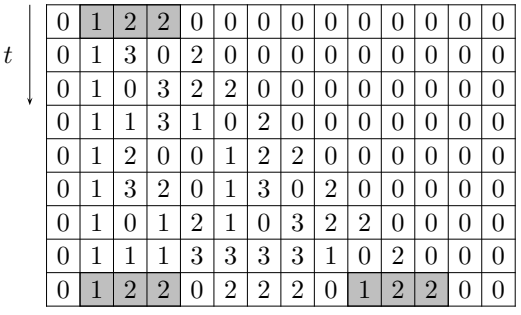
Roughly speaking, "to be contained in a quiescent environment" means that $\sup(\Delta^t(c))$ contains nothing else than the copies of c.

Exemplarily we consider dimension one. The theorem is witnessed by the cellular automaton $\langle S, 1, (-1, 0), \delta, 0\rangle$, where $S = \{0, 1, \ldots, n-1\}$ for $n \geq 2$, and δ is the addition modulo n, that is $\delta(s_1, s_2) = (s_1 + s_2) \bmod n$. Let c be a configuration with $\sup(c) = \{0, 1, \ldots, k\}$. Then by induction one shows that

$$\Delta^r(c)(i) = \left(\sum_{x=0}^{i} \binom{r}{i-x} c(x)\right) \bmod n$$

for $i \in \{0, 1, \ldots, k\}$ and $r \geq 0$. Setting $r = n \cdot k!$ and $0 \leq x < i \leq k$ we obtain that n divides $\binom{r}{i-x}$. Therefore, the sum (mod n) is equal to $\binom{r}{0} c(i) \bmod n = c(i)$, and we have $\Delta^{n \cdot k!}(c)(i) = c(i)$, for $0 \leq i \leq k$. Similarly, it is shown that $\Delta^{n \cdot k!}(c)(i + n \cdot k!) = c(i)$, for $0 \leq i \leq k$. It follows that the configuration $\Delta^{n \cdot k!}(c)$ contains two disjoint subconfigurations which are equal to c. One subconfiguration is exactly c, the other one is translated $n \cdot k!$ cells to the right.

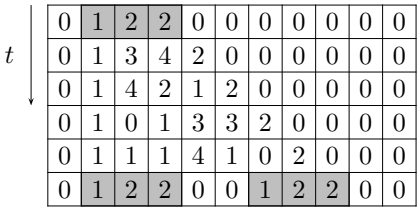
Example 3. $S = \{0, 1, 2, 3\}$, $n = 4$, $k = 2$, and $r = n \cdot k! = 8$. The embedded pattern 122 is reproduced after eight steps as depicted in the following space-time diagram.



□

If the number n of states in S is a prime number then r can be chosen to be the smallest power of n not less than k. This follows from the fact that for prime numbers p each term $\binom{p^r}{x}$ with $1 \leq x < p^r$, for any $r \geq 1$, is of the form $p^r \cdot y$ for some integer y, hence divisible by p. Since, in particular, divisibility by p means to be equal zero modulo p and zero is the quiescent state, in this case the copies of the initial pattern are contained in a quiescent environment.

Example 4. $S = \{0, 1, 2, 3, 4\}$, $n = 5$, $k = 2$, and $r = 5^1 \geq 2$. The embedded pattern 122 is reproduced in a quiescent environment after five steps as depicted in the following space-time diagram.



□

7 Concluding Remarks

We reviewed the beginnings of and some basic aspects concerning nontrivial artificial self-seproduction, in particular in connection with the model of cellular automata. Von Neumann's self-reproducing universal computer-constructor was addressed. Then we turned to the (still unanswered) question what *nontrivial* self-reproduction is. The question is not precise and admits to trivial versions as well as interesting ones. So, the question as to what kind of logical organization is *necessary* for an artificial structure to be able to reproduce itself arises immediately. Inspired by these ideas and results, further issues concerning artificial structures and self-reproduction are addressed. The presentation obviously

lacks completeness. There is a vast of literature dealing with the problems in question. We emphasize a result by Morita and Imai [12] which shows that self-reproduction is also possible in *reversible* cellular automata. For a comprehensive survey covering "Fifty Years of Research on Self-Replication" we refer to [14].

References

1. Amoroso, S., Cooper, G.: Tesselation structures for reproduction of arbitrary patterns. J. Comput. System Sci. 5, 455–464 (1971)
2. Arbib, M.A.: Simple self-reproducing universal automata. Inform. Control 9, 177–189 (1966)
3. Banks, E.R.: Information processing and transmission in cellular automata. Technical Report MAC TR-81, MIT (1971)
4. Berlekamp, E.R., Conway, J.H., Guy, R.K.: What is Life? In: Winning Ways for your Mathematical Plays?, vol. 2, ch. 25, pp. 817–850. Academic Press, London (1982)
5. Burks, A.W.: Von Neumann's self-reproducing automata. In: Burks, A.W. (ed.) Essays on Cellular Automata, pp. 3–64. University of Illinois Press, Urbana (1970)
6. Codd, E.F.: Cellular Automata. Academic Press, New York (1968)
7. Gardner, M.: The fantastic combinations of John Conway's new solitaire game 'life'. Sci. Amer. 223, 120–123 (1970)
8. Hamilton, W.L., Mertens Jr., J.R.: Reproduction in tesselation structures. J. Comput. System Sci. 10, 248–252 (1975)
9. Herman, G.T.: On universal computer-constructors. Inform. Process. Lett. 2, 61–64 (1973)
10. Langton, C.G.: Self-reproduction in cellular automata. Phys. D 10, 135–144 (1984)
11. Moore, E.F.: Machine models of self-reproduction. Proc. Symposia in Applied Mathematics 14, 17–33 (1962)
12. Morita, K., Imai, K.: Self-reproduction in a reversible cellular space. Theoret. Comput. Sci. 168, 337–366 (1996)
13. Ostrand, T.J.: Pattern reproduction in tesselation automata of arbitrary dimension. J. Comput. System Sci. 5, 623–628 (1971)
14. Sipper, M.: Fifty years of research on self-replication: An overview. Artificial Life 4, 237–257 (1998)
15. Thatcher, J.W.: Universality in the von neumann cellular model. Technical Report 06376, 06689, 03105-30-T, University of Michigan (1964)
16. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc., Series 2 42, 230–265 (1936)
17. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc., Series 2 43, 544–546 (1937)
18. von Neumann, J.: Theory of Self-Reproducing Automata. University of Illinois Press, Urbana (1966)
19. Wainwright, R.T.: Life is universal!. In: Winter Simulation Conference (WSC 1974). WSC/SIGSIM, vol. 2, pp. 449–459 (1974)

An Algorithmic Approach to Tilings of Hyperbolic Spaces: 10 Years Later

Maurice Margenstern

Université Paul Verlaine – Metz,
LITA EA 3097, UFR MIM, and CNRS, LORIA,
Campus du Saulcy,
57045 METZ Cédex 1, France
margens@univ-metz.fr
http://www.lita.sciences.univ-metz.fr/~margens

Abstract. In this paper, we give an account of the algorithmic approach developed by the author to study tilings of hyperbolic spaces. We sketchily remember the results which were obtained by this approach and we conclude by possible applications, indicating a few ones already performed and proposing three others.

Introduction

Ten years ago, a paper appeared in the *Journal of Universal Computer Science* which laid the basis of an algorithmic approach to tilings of the hyperbolic plane. This approach appeared to be very fruitful and it gave rise to around 85 papers in journals, conferences and workshops. A few other papers also appeared connected with other approaches to such tilings, motivated by a combinatorial point of view too.

In this paper, we give an account of the works of the algorithmic approach developed by the author and to possible applications.

In Section 1, the paper will remind Poincaré's disc model which summarizes what is needed to know from hyperbolic geometry in order to understand the results which will be presented. In Section 2, we shall explain our approach on the classical case of the pentagrid, the tiling $\{5, 4\}$ of the hyperbolic plane which is obtained from the regular rectangular pentagon by reflection in its sides and, recursively, by reflection of the images in their sides. We shall see that the tiling is spanned by a tree and that numbering the nodes of the tree in an appropriate way together with a suited representation of the numbers will make an important property appear from which we can easily derive **navigation tools** in the tiling. In Section 3, we show a remarkable property of the hyperbolic plane: the navigation tools obtained for the pentagrid work exactly in the same way in the heptagrid, the tiling $\{7, 3\}$ of the hyperbolic plane, and, with a simple adaptation, to all tilings of the hyperbolic plane of the form $\{p, 4\}$ and $\{p+2, 3\}$. We also indicate there the possible generalizations to other tilings of the hyperbolic plane as well as of higher dimension, namely dimensions 3 and 4.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 37–52, 2010.

In Section 4, we remind the undecidability results obtained about tilings. This concerns the tiling problem itself as well as connected results.

In Section 5, we consider the application of the navigation tools to the implementation of cellular automata in hyperbolic tilings. The obtained results can be divided into three categories: general properties, complexity results and universality results. We shall briefly mention the universality results which were investigated for the pentagrid, the heptagrid and the dodecagrid. This latter tiling is the tessellation of the hyperbolic $3D$ space obtained from what is called Poincaré's dodecahedron whose faces are copies of the regular rectangular pentagon. Figure 4 represents the basic bricks which allow us to construct this tilings. All universality results which will be presented will be based on the implementation of a railway circuit in the considered hyperbolic grid.

In Section 6, we investigate possible applications of the navigation tools. In a first sub-section, we shall look at already obtained applications. Then, in the second sub-section, we shall look at possible applications to biology. One application is theoretical and deals with an implementation of P systems, it was already presented a few years ago at the occasion of an issue of WMC. The other two applications are more practical. The paper just gives a proposal which would require a rather huge effort.

1 Poincaré's Disc

The points of the hyperbolic plane are the points of the open unit disc U whose border, ∂U, consists of the **points at infinity**. These latter points are represented by the circle of Figure 1. Note that the points at infinity do not

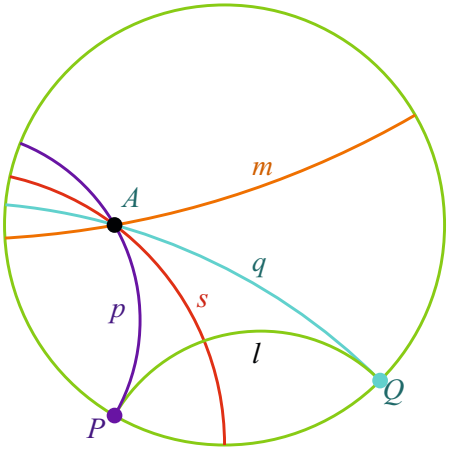


Fig. 1. Illustration of Poincaré's disc model

belong to the hyperbolic plane. Lines are trace on the open disc of diameters or circles which are orthogonal to ∂U. Two lines are parallel if and only if they have a common point at infinity. In Figure 1, we can see two parallels to ℓ which pass through A. We can see also a line passing through A which cuts ℓ. Now there is another kind of lines: in Figure 1, line m which also passes through A does not cut ℓ, neither in U, nor on ∂U, nor outside U. Such lines are called **non-secant** lines. They are characterized by the following property: they possess a unique common perpendicular.

2 The Pentagrid

The pentagrid is one of the infinitely many tessellations which live in the hyperbolic plane. Remember that a **tessellation** is defined starting from a convex polygon P replicated by reflection in its sides and, recursively, by replicating the images through reflections in their sides. We say that P generates a tessellation if the images cover all the plane without overlapping. A theorem of Poincaré says that any triangle with $\frac{2\pi}{p}$, $\frac{2\pi}{q}$ and $\frac{2\pi}{r}$ as angles, where p, q and r are positive integers satisfying $\frac{1}{p}+\frac{1}{q}+\frac{1}{r}<\frac{1}{2}$ generates a tessellation.

Accordingly, there are infinitely many tessellations in the hyperbolic plane. If we consider a regular polygon P with p sides and $\frac{2\pi}{q}$ as angle, P generates a tessellation if and only if $\frac{1}{p}+\frac{1}{q}<\frac{1}{2}$. The tessellation is denoted by $\{p,q\}$. We can interpret q as the number of copies of P which can be put around a vertex filling a neighbourhood of the vertex without overlapping.

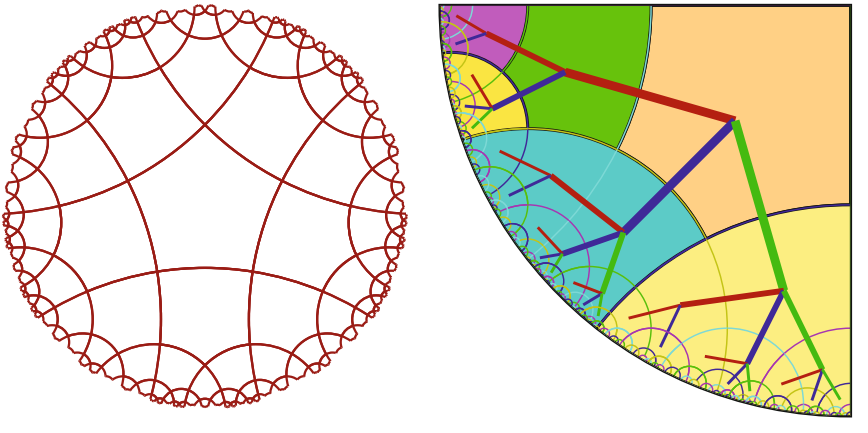


Fig. 2. The pentagrid and its underlying tree: the nodes of the tree are in bijection with the tiles which exactly cover a quarter of the plane

The pentagrid is one of these tessellations, namely $\{5, 4\}$. It is generated by the regular pentagon with right angles, whence the name. Note that there cannot be a rectangular polygon with a smaller number of sizes in the hyperbolic plane.

As shown by Figure 2, the pentagrid is spanned by a tree. Figure 2 also summarizes the way it is generated by an appropriate splitting of the hyperbolic plane. We cannot describe this splitting here, referring the reader to [12,21]. What we can say is that the tree represented in Figure 2 is not arbitrary. It has two kinds of nodes called **black** and **white** with two and three sons respectively. In both cases, exactly one son is black. From that it is not difficult to prove that the number of nodes which are on the same level of the tree belongs to the Fibonacci sequence. More precisely, if h is the level, the number of nodes at this level is f_{2h+1} where $\{f_n\}_{n\in N}$ with $f_0 = f_1 = 1$ is the usual Fibonacci sequence. This is why we called this tree the **Fibonacci tree**. Now, number the nodes level by level from the root of the tree and, on each level, from left to right. Represent each number as a sum of single terms of the Fibonacci sequence, always taking the longest representation, as there my be several representations. Call **coordinate** of a tile this representation. Then, we have:

Theorem 1. *(Margenstern,* 2002*, [13,21]) The language of the coordinates for the tiles of the pentagrid is regular.*

Theorem 2. *(Margenstern,* 2000*, [12,15,21]) There is an algorithm to compute the path from the root to a node from its coordinate which is linear in time in the size of the coordinate.*

3 Generalizations

As indicated by the title of this section, these properties are not specific for the pentagrid. In the following sub-sections, we shall see several successive generalizations.

3.1 The Heptagrid and the Tilings $\{p, 4\}$ and $\{p+2, 3\}$, $p \geq 5$

The first striking generalization is given by the transposition of the previous theorems to the heptagrid, the tiling $\{7, 3\}$ of the hyperbolic plane. This tiling is illustrated by Figure 3 in which we also can see the underlying tree structure.

In Figure 3 we can not only see that the heptagrid is spanned by a tree but also that the tree is the Fibonacci tree, exactly as it is for the pentagrid. Here, the tree does not span a quarter of the plane but an angular sector corresponding to the angle $\frac{2\pi}{7}$, as we can see on Figure 5. Accordingly, theorems 1 and 2 also hold for the heptagrid.

This identity of the spanning tree raised the question of how we can go from one tiling to the other? This issue was solved in [18,21] where another system of coordinates based on the same tree is proposed. The construction of [18,21] allows us to implement a bi-continuous bijection between the pentagrid and the heptagrid.

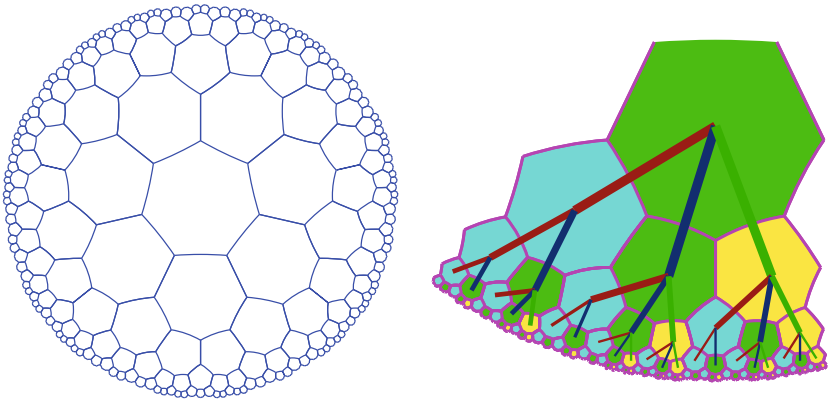


Fig. 3. The underlying tree of the tiling: the nodes of the tree are in bijection with the tiles which exactly cover an angular sector of the plane

The reader is referred to [4,21] for a detailed account on the splitting performed in the heptagrid.

3.2 The Splitting Method and the Tilings $\{p, q\}$

In fact, the property of being spanned by a tree is shared by almost all tessellations of the hyperbolic plane. This result was proved by the author in [13,14] where the **splitting method** was introduced. This method defines the notion of a **basis of a splitting**. Informally, this means that the geometric space under consideration can be generated by finitely many **prototiles**, and that the space can be split into finitely many regions, each one being defined as a union of copies of the prototiles. In the case of the pentagrid and of the heptagrid, there is one prototile and two regions. We refer the reader to the already quoted papers and to the book [21]. When such a basis exists, we say that the tiling is **combinatoric** and the existence of the basis immediately yields a matrix whose characteristic polynomial contains the information giving rise to the Fibonacci representation in the case of the pentagrid and of the heptagrid.

Theorems 1 and 2 where first extended to the tilings $\{p, 4\}$, see [35] and then to the tilings $\{p+2, 3\}$, with $p \geq 5$, see [21]. They were also extended to many tilings $\{p, q\}$, see [21] where the difficult case is given by the triangular tilings, which was already noticed in [14].

3.3 The Dodecagrid and the 120-Grid

Another direction consists in extending the method to higher dimensions. This was performed in [36] and [17].

The Poincaré's disc model naturally extends to higher dimensions: replace the open disc by the n-dimensional open hyper-ball, the circle by the n-dimensional hyper-sphere, the lines by diametral hyperplanes or n-dimensional hyper-spheres orthogonal to the border of the fixed n-dimensional open hyper-ball. Other k-dimensional objects, with $k < n-1$ are defined as intersections of $k+1$-dimensional ones. A theorem by Sommerville shows that starting from dimension 5, there are no tessellations in hyperbolic spaces. In dimensions 3 and 4 for there are only finitely many of them: 4 and 5 respectively. This is a sharp difference with the situation in the plane.

In the hyperbolic $3D$ space, the **dodecagrid** is the natural extension of the pentagrid: as the cubic tessellation is the natural extension of the planar square grid in the Euclidean case. This tiling was proved to be combinatoric in [36], and Figure 4 gives a new basis of splitting, a bit simpler than the one indicated in [36,21].

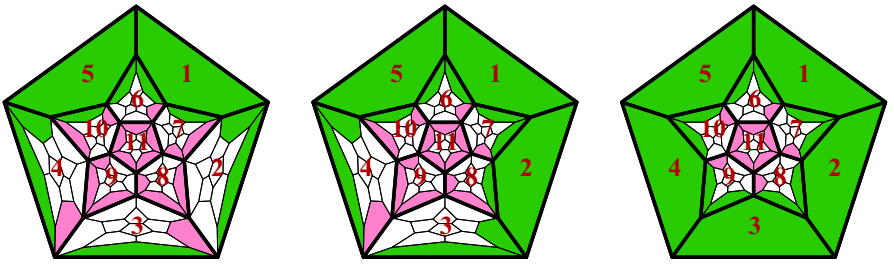


Fig. 4. The lego-like bricks for the dodecagrid

The extension of the dodecagrid to the hyperbolic $4D$ space exists and it gives rise to a tessellation called the **120-grid** after the **120-cell polytope** which is the polytope on which the tessellation is performed. The author proved that this tiling also is combinatoric, see [17,21].

4 The Tiling Problem

Consider a **finitely generated** tiling. This means that we have a finite set of **prototiles** and that the space is a union of **copies** of the prototiles without overlapping. By **copy** of a prototile, we mean an image of a prototile under an isometry of the space belonging to a fixed set of such transformations. We may add **decorations** on the prototiles: this means that each side or facet is assigned a symbol and that when a facet is shared by two copies, the facet must bare the same symbol in each copy. We call this the **matching condition at a facet**. In this case, a solution of the tiling problem is an assignment of the copies so that the matching condition is satisfied at each facet.

The tiling problem consists in the following question:

Is there an algorithm which, from a description of a finite set of tiles, says yes if there is a solution and no if such a solution does not exist?

This problem was raised by Hao Wang in 1958 for the Euclidean plane and this author proved a partial case to be undecidable. This partial case is called the origin constrained problem: the first tile is fixed in advance. The general problem, with no constraint, was proved undecidable by Robert Berger in 1966, see [2]. The complex, but deep proof of Berger was significantly simplified in 1971 by Raphael Robinson, see [43] who raised the question of the same problem for the hyperbolic plane. In 1978, Robinson proved the undecidability of the origin constrained problem in the hyperbolic plane, see [44]. The algorithmic approach presented in this paper allowed the author to prove the undecidability of the general problem in the hyperbolic plane in 2007. The complete proof was published in 2008, see [23]. A synthetic view of the proof appeared in 2007 in [22]. It is impossible, within the frame of this paper to go into details. The main goal is to implement infinitely many simulations of the same Turing machine starting from an empty tape. The problem boils down to implement large enough portions of grids for this simulation and also in a rather uniform way in order to force the construction. The main idea was to define a family of trees which are pairwise either disjoint or one of them is embedded in the other. Then, we cut these trees at appropriate levels defining triangle like figures of infinitely many different sizes. From the legs of these triangles, extremal branches of the trees, we define verticals v_i which avoid the triangles which are inside the triangle where the v_i were defined. This defines the grids in which the above mentioned simulation takes place. The reader who wishes to know how these trees and triangles are implemented is referred to [22,23].

A bit later, two other problems about tiling, also connected to this one were proved undecidable by the author: the finite tiling problem, see [24], and the periodic tiling problem, see [29].

5 Cellular Automata in Hyperbolic Spaces

An important application of this algorithmic approach to hyperbolic geometry is the implementation of cellular automata in the hyperbolic plane. The global setting is illustrated by Figure 5. In each figure, α sectors are displayed around a central tile with $\alpha = 5$ for the pentagrid and $\alpha = 7$ for the heptagrid. This is an exact covering: in Figure 5, the sectors are a bit split away from each other and from the central tile to emphasize on the shape of the sectors. In the pentagrid, the sectors are quarters of the plane, as we already know. In the heptagrid, they are angular sectors defined by two rays with an appropriate angle. These rays are supported by so-called **mid-point lines** as their supporting lines pass through the mid-points of consecutive edges of the heptagons they cross.

Now, we can define coordinates as follows: 0 is attached to the central cell. Then, we number the sectors by counter-clockwise turning around the central tile, fixing the first sector once and for all. Next, in each sector σ, with $\sigma \in \{1..\alpha\}$, the tiling is in bijection with the Fibonacci tree. Accordingly each tile takes the coordinate ν of the node to which it corresponds in the tree. And so, except the central tile, the coordinate of a tile is given by the two numbers ν and σ and we

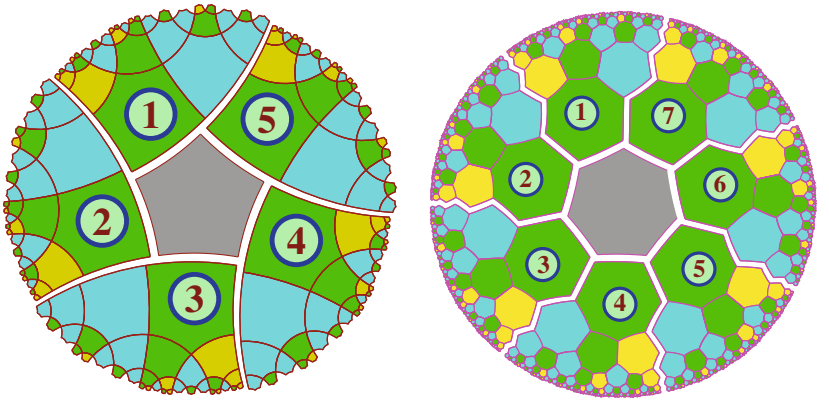


Fig. 5. Left-hand side: setting for the pentagrid. Right-hand side: setting for the heptagrid.

write $\nu(\sigma)$. We also attach to each cell a **local numbering** which consists in numbering the sides of the tile from 1 to α by counter-clockwise turning around the tile. In the central cell, side 1 is the side in contact with the first sector. In the other cells, side 1 is the side in contact with the father of the tile in the tree. We say that the cell which is on the other side of side i is the neighbour i, so that neighbour 1 is the father of the cell. Note that we define the father of the root to be the central cell. This allows us to gather the trees spanning the sectors as sub-trees of a bigger tree rooted at the central cell.

This allows us to define the format of the rules. We write them as follows: $\eta_0\eta_1..\eta_\alpha \to \eta_0^1$ or simpler $\eta_0\eta_1..\eta_\alpha\eta_0^1$. In these notations, η_0 is the current state of the cell, η_i is the state of its neighbour i and η_0^1 is the state of the cell after applying the rule to the cell. We also say that η_0^1 is the **new state** of the cell.

Note that this implementation corresponds to what is performed in the square grid of the Euclidean plane identified with $\mathbb{Z}^2$. This identification consists of three steps. First, we define a square to correspond to $(0, 0)$. Then, we define which neighbour of the square through a side will be called north, which fixes the square attached to $(0, 1)$. Next, the orientation, which does not belong to the Euclidean plane itself, consists in fixing which square will be attached to $(1, 0)$. We have an analogous situation in the pentagrid. First, we fix the central cell. Then, we fix which neighbour will be neighbour 1. At last, orientation, it does not belong neither to the hyperbolic plane, allows us to fix the leftmost son of the roots.

An important case of cellular automata in the hyperbolic plane is the case of **rotation invariant** cellular automata. For these automata, the set of rules has the property that if $\eta_0\eta_1..\eta_\alpha\eta_0^1$ belongs to the set, all rules $\eta_0\eta_{\pi(1)}..\eta_{\pi(\alpha)}\eta_0^1$ belong to the set too, where π is any circular permutation on $[1..\alpha]$. This means that the rules are the same if a rotation is performed around a cell. In particular, for rotation invariant cellular automata, the rules are independent of the way we fix side 1 in the cells. Only the ordering corresponding to a rotation around the tile is required.

5.1 General Results

An important notion, in the mathematical study of cellular automata, is the **global function** attached to the cellular automaton. Let A be a cellular automaton. We can denote by f_A the function defined by the set of rules. If Q denotes the set of states of A, f_A is a mapping from $Q^{\alpha+1}$ into Q. A **configuration** is an assignment of the states to each cell. Assuming that the cells are numbered in a bijective way by $\boldsymbol{N}$, a configuration is a mapping from $\boldsymbol{N}$ into Q. Let X denote the set of configurations. The global function G_A of A is a mapping of X into itself defined by: $G_A(x)(c) = f_A(c, V(c))$, where $x \in X$, c is a cell and $V(c)$ is the set of neighbours of c, ordered from 1 to α.

In the Euclidean case, properties of the global function of a cellular automaton were investigated. In particular, a characterization of the mappings from X into itself which are the global function of some cellular automaton was found, see [6]. As a non-trivial corollary, if the global function of a cellular automaton A is bijective, then the reverse function is also the global function of another cellular automaton called the inverse of A. However, it is worth noticing that the proof of this latter property is not constructive. Also, an interesting connection between the injectivity of the global function and its surjectivity were found, see [42,41]. However, these properties were proved to be undecidable as the injectivity of the global function was proved undecidable, see [11].

The author was able to prove a few analogues of these theorems for hyperbolic cellular automata with, however, differences with the Euclidean situation. As an example, Hedlund's characterization for the global function of a cellular automaton can be transported for rotation invariant cellular automata only, see [25]. The reason is that Hedlund's characterization involves shifts and that in the hyperbolic plane, shifts generate rotations. However, the connections between injectivity and surjectivity of the global function do not hold for hyperbolic cellular automata, see [28]. At last, the injectivity of the global function of a hyperbolic cellular automaton was also proved undecidable, see [27]. In the Euclidean case, the proof starts from the construction used to prove the undecidability of the tiling problem and it mixes this construction with the construction of a path filling up the plane. In the hyperbolic case, the proof mixes the construction used to prove the undecidability of the tiling problem with the construction of a path which fills up at least a large enough region of the plane.

5.2 Complexity Results

Closer to the core of computer science, the complexity classes of cellular automata in the hyperbolic plane turned out to have surprising properties. The very first result was a proof that **SAT** can be solved in polynomial time, see [34]. The authors did not know at that time that the property was foreseen in [40] where, however, no precise description in terms of cellular automata was given. A bit later, it was proved that $\mathbf{P}_h = \mathbf{NP}_h$, see [10,26], where the subscript h refers to the cellular automata in the hyperbolic plane. It is important to notice that the results deals with time only, with no assumption on the space used for

the computation. It turned out that the class $\mathbf{P}_h$ is very large as it is the same as the classical **PSPACE**. It was also proved that $\mathbf{EXP}_h$, again the purely time complexity class, is as large as **EXPSPACE**, see [9].

The result about **SAT** relies on the fact that a complete binary tree of height h can be constructed in the pentagrid in time h. In fact, the construction of [34] is not only polynomial in time, it is linear. Now, the fact that this construction can easily and efficiently be performed in the hyperbolic plane has two consequences. It seems to me that this points at the fact that the traditional question is **P** equal to **NP**? is an ill posed problem: it was not correct to ignore any connection with space while raising the question. But at the same time, it seems to me that the result that $\mathbf{P}_h = \mathbf{NP}_h$ is an important clue in favour of the statement that $\mathbf{P} \neq \mathbf{NP}$. A way of attacking this issue could be the following: if $\mathbf{P} = \mathbf{NP}$ were true, it would be possible to embed the hyperbolic plane into the Euclidean one while preserving both angles and distances. Now, it is known that this latter possibility is ruled out.

5.3 Universality Results

Another trend of research for cellular automata is the search of universal cellular automata with as less states as possible. Also, for this purpose, very soon people focused on **weak** universality. This means the following. We have a **quiescent** state q characterized by the fact that if a cell is under the state q, if its neighbours are also all under this state, the new state of the cell is again q. In this context, we say that a configuration is **finite** if all cells except finitely many of them are in the same quiescent state. **Strong** universality means that a Turing machine can be simulated by the cellular automaton, when this latter one starts from a finite initial configuration. Weak universality means that we relax the condition of finite initial configuration. However, arbitrary initial configurations are also not allowed, in which case the cellular automaton could solve the halting problem, for instance. Usually, it is required that the initial configuration is regular is some sense to be closer indicated. In what follows, universality always means weak universality.

The first universal cellular automaton in the hyperbolic plane appeared in [8]. This cellular automaton had 22 states and it simulated the computation of a two-register machine through the motion of a locomotive on a railway circuit, adapting the implementation in the Euclidean plane devised by [45] to the hyperbolic plane, also replacing the simulation of a Turing machine by the simulation of a register machine. There was also an implementation of the same railway circuit in the dodecagrid, this time with 5 states, see [20]. A few years later, the number of states was lowered down to 9 in the pentagrid, see [37]. Then a simulation in the heptagrid was performed with 6 states, see [38]. This latter result was recently lowered down to 4 in the heptagrid, see [30]. The simulation in the dodecagrid was also lowered down to 3, see [31]. Very recently, the simulation in the dodecagrid was lowered down to 2, so that this result cannot be improved, at least for what is the number of states. Also very recently, a universal cellular

automaton with two states was found in the heptagrid and in the pentagrid by implementing a universal cellular automaton on the line called rule 110. See [5] for rule 110 and [32] for the hyperbolic plane.

6 Possible Applications

The previous sections have already shown several applications of the algorithmic approach to hyperbolic geometry. Of course, these are theoretical applications: applications to mathematics and to computer science.

In the following sub-section, we shall deal with more *practical* applications. In Sub-section 6.1, we look at those already performed. In Sub-section 6.2, we suggest a few ones which might be possible.

6.1 What Was Already Performed

Two applications were already performed. One deals with a colour chooser. The other deals with keyboards for various mobile devices like cell phones, ipods and many others. All these applications are based on the possibility to move a window over the pentagrid. To ensure this possibility, we have an algorithm to know the new coordinates of a tile when the system of coordinate is changed.

The problem is the following. We have a fixed system of coordinate, the one already defined in Section 5. We call it the **absolute** system. Now, we consider the centre of the window as the centre of a coordinate system again, which we call the **relative** system. And so, knowing the absolute coordinate of the centre of the window, it is not very difficult to compute the absolute coordinates of the tiles which can be seen in the window, see [19,26]. We call this the **recomputation algorithm**.

The Colour Chooser. The colour chooser is based on the heptagrid. It could also be based on the pentagrid. However, the heptagrid provides a better result, both for efficiency and from an aesthetic point of view.

The principle of the chooser is the following. We fix a central cell, the black one in Figure 6, which plays the role of the absolute system. Each tile receives a colour which is defined by its radial position and its distance from the centre. A first approximation of the radial position is given by the number of the sector to which belongs the considered tile τ. Next, the coordinate of τ in its tree allows us to determine the level to which it belongs, hence the distance to the centre, and also to know the position of the tile on the corresponding level of the tree. The radial position defines the hue of the colour and the distance from the center defines its intensity, all these parameters computed in the interval $(0, 1)$, using the HSB system. The third parameter is always set to 1.

If the user wishes to select a colour, he/she moves in the heptagrid until it finds the required colour. The motion is produced by placing the centre of the window on the tile selected by the user. The recomputation algorithm allows us

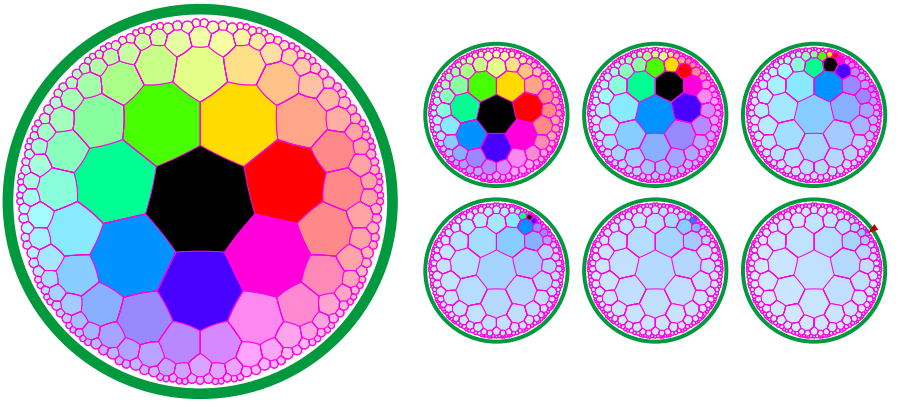


Fig. 6. The colour chooser. On the right-hand side several moves. On two of them, the initial centre disappeared. It is indicated by an appropriate arrow in the third figure. The arrow works like a **compass**. Without it, once the old centre vanished from the window, the user could never put it again inside the window.

to compute the absolute coordinates of the tiles around the centre of the window and so, to produce the correct colour of the considered tile. We refer the reader to [3] for more details on the chooser.

The Keyboards. As indicated in the beginning of this section, another application deals with keyboards for mobile devices where there is no convenient room for a standard keyboard. The first realization was for a French keyboard using the pentagrid, see [39]. Then, it came to the author that the pentagrid is especially suited for the Japanese language based on five vowels. This leads to the presentation displayed by Figure 7 on which we can also see a prototype of a true cell phone using this idea.

The idea is simple. Using the traditional presentation of the syllabic alphabets in the Japanese language, a quarter around the central cell is attached to each vowel and the syllables containing it are displayed around the root-tile following the traditional order. The reader will find more information in [33].

6.2 What Could Be Done

Many applications are possible. An already quoted paper, [19], is also a basis for applications to computer science which we have not the room to present, ranging from the Internet to data storage and queries.

Here, we would like to briefly mention two new possible applications to biology. A rather old one was presented for the representation of P systems, see [16].

Among the new applications, the first one deals with the brains and the other with diffusion processes.

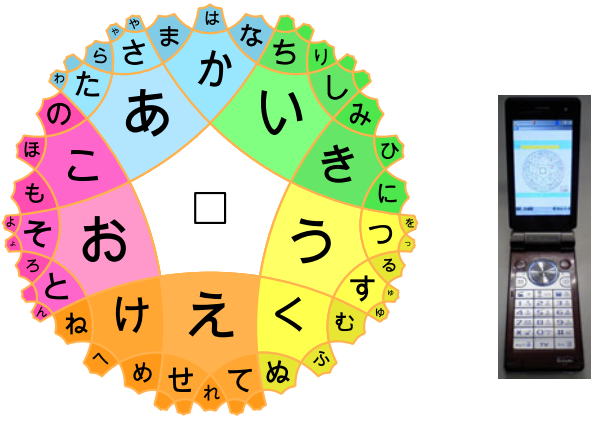


Fig. 7. The Japanese keyboard. Left-hand side: hiraganas, right-hand side: cell phone prototype.

Fig. 8. A crochet representation of the hyperbolic plane by Daina Taimina

The Brains. This idea comes from a rather well known fact that if we implement a tiling of the hyperbolic plane while keeping the lengths and the angle on a soft textile medium, the surface cannot remain straight and it folds itself more and more as the implemented portion of the hyperbolic plane is growing. A striking example of this is given by the crochet realizations by Daina Taimina, see [7]. Here is one of them:

This reminds the circumvolutions of the brains. Hyperbolic tilings could give a possible new way to map the regions controlled by the nervous system.

Diffusion Processes. There are many diffusion processes in biology, starting with the growth of colonies of bacteria. Any simulation of these processes, as the

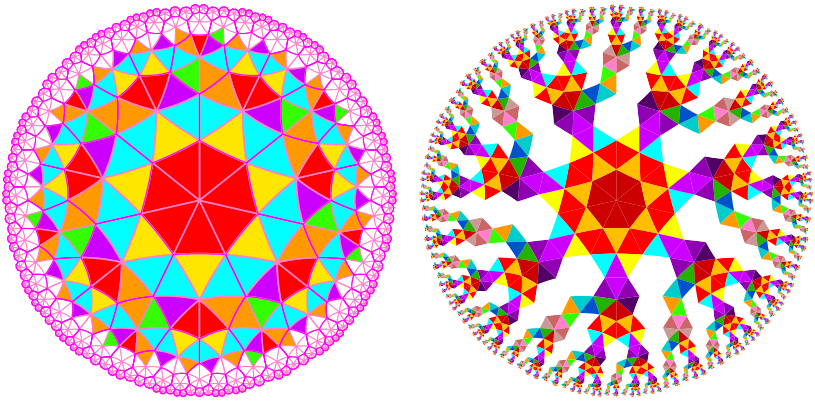


Fig. 9. Left-hand side: the 7^{th} step of the diffusion of the tree structure in a triangular subdivision of the heptagrid. Right-hand side: a similar diffusion at its 35^{th} step, in a finer subdivision of the previous one. Compare with Figure 2 in [1].

ones of Figure 9 could contribute to a better knowledge of these phenomena. It is now well established that colonies of bacteria have enormous abilities to adapt themselves to environment, even to very hostile ones, see [1].

Figure 9 illustrates the propagation of the tree structure by a cellular automaton. On the left-hand side, the cells are triangles defined by the side and the centre of a heptagon. On the right-hand side, the cells are triangles obtained by splitting the previous ones into four sub-triangles, using the mid-points of their sides.

Acknowledgment

The author is much in debt to the organizers of CMC'11 for inviting him to deliver a talk of which the following paper is a very concise summary.

References

1. Ben-Jacob, E., Becker, I., Shapira, Y.: Bacterial Linguistic Communication and Social Intelligence. Trends in Microbiology 12(8), 366–372 (2004)
2. Berger, R.: The undecidability of the domino problem. Memoirs of the American Mathematical Society 66, 1–72 (1966)
3. Chelghoum, K., Margenstern, M., Martin, B., Pecci, I.: Palette hyperbolique: un outil pour interagir avec des ensembles de données. In: IHM 2004, Namur (2004) (Hyperbolic chooser: a tool to interact with data, (French))
4. Chelghoum, K., Margenstern, M., Martin, B., Pecci, I.: Tools for implementing cellular automata in grid $\{7, 3\}$ of the hyperbolic plane. In: DMCS 2004 (2004)
5. Cook, M.: Universality in Elementary Cellular Automata. Complex Systems 15(1), 1–40 (2004)

6. Hedlund, G.: Endomorphisms and automorphisms of shift dynamical systems. Math. Systems Theory 3, 320–375 (1969)
7. Henderson, D.W., Taimina, D.: Crocheting the Hyperbolic Plane. Mathematical Intelligencer 23(2), 17–28 (2001)
8. Herrmann, F., Margenstern, M.: A universal cellular automaton in the hyperbolic plane. Theoretical Computer Science 296, 327–364 (2003)
9. Iwamoto, C., Margenstern, M.: Time and Space Complexity Classes of Hyperbolic Cellular Automata. IEICE Transactions on Information and Systems 387-D(3), 700–707 (2004)
10. Iwamoto, C., Margenstern, M., Morita, K., Worsch, T.: Polynomial Time Cellular Automata in the Hyperbolic Plane Accept Exactly the PSPACE Languages. In: SCI 2002 (2002)
11. Kari, J.: Reversibility and Surjectivity Problems of Cellular Automata. Journal of Computer and System Sciences 48(1), 149–182 (1994)
12. Margenstern, M.: New Tools for Cellular Automata of the Hyperbolic Plane. Journal of Universal Computer Science 6(12), 1226–1252 (2000)
13. Margenstern, M.: A contribution of computer science to the combinatorial approach to hyperbolic geometry. In: SCI 2002 (2002)
14. Margenstern, M.: Revisiting Poincaré's theorem with the splitting method. In: Bolyai 200. International Conference on Hyperbolic Geometry, Cluj-Napoca, Romania, October 1-4 (2002)
15. Margenstern, M.: Implementing Cellular Automata on the Triangular Grids of the Hyperbolic Plane for New Simulation Tools. In: ASTC 2003 (2003)
16. Margenstern, M.: Can Hyperbolic Geometry Be of Help for P Systems? In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 240–249. Springer, Heidelberg (2004)
17. Margenstern, M.: The tiling of the hyperbolic $4D$ space by the 120-cell is combinatoric. Journal of Universal Computer Science 10(9), 1212–1238 (2004)
18. Margenstern, M.: A new way to implement cellular automata on the penta- and heptagrids. Journal of Cellular Automata 1(1), 1–24 (2006)
19. Margenstern, M.: On the communication between cells of a cellular automaton on the penta- and heptagrids of the hyperbolic plane. Journal of Cellular Automata 1(3), 213–232 (2006)
20. Margenstern, M.: A universal cellular automaton with five states in the 3D hyperbolic space. Journal of Cellular Automata 1(4), 315–351 (2006)
21. Margenstern, M.: Cellular Automata in Hyperbolic Spaces. Theory, vol. 1, p. 422. Old City Publishing, Philadelphia (2007)
22. Margenstern, M.: The Domino Problem of the Hyperbolic Plane is Undecidable. Bulletin of the EATCS 93, 220–237 (2007)
23. Margenstern, M.: The domino problem of the hyperbolic plane is undecidable. Theoretical Computer Science 407, 29–84 (2008)
24. Margenstern, M.: The Finite Tiling Problem Is Undecidable in the Hyperbolic Plane. International Journal of Foundations of Computer Science 19(4), 971–982 (2008)
25. Margenstern, M.: On a Characteriztion of Cellular Automata in Tilings of the Hyperbolic Plane. International Journal of Foundations of Computer Science 19(5), 1235–1257 (2008)
26. Margenstern, M.: Cellular Automata in Hyperbolic Spaces. Implementation and computations, vol. 2, p. 360. Old City Publishing, Philadelphia (2008)

27. Margenstern, M.: The Injectivity of the Global Function of a Cellular Automaton in the Hyperbolic Plane is Undecidable. Fundamenta Informaticae 94(1), 63–99 (2009)
28. Margenstern, M.: About the Garden of Eden theorems for cellular automata in the hyperbolic plane. Electronic Notes in Theoretical Computer Science 252, 93–102 (2009)
29. Margenstern, M.: The periodic domino problem is undecidable in the hyperbolic plane. In: Bournez, O., Potapov, I. (eds.) RP 2009. LNCS, vol. 5797, pp. 154–165. Springer, Heidelberg (2009)
30. Margenstern, M.: A universal cellular automaton on the heptagrid of the hyperbolic plane with four states. Theoretical Computer Science (to appear, 2010)
31. Margenstern, M.: A weakly universal cellular automaton in the hyperbolic $3D$ space with three states. arXiv:1002.4290v1[cs,DS], p. 54
32. Margenstern, M.: A weakly universal cellular automaton in the hyperbolic $3D$ space with two states. arXiv:1005.4826v1[cs,FL], p. 38
33. Margenstern, M., Martin, B., Umeo, H., Yamano, S., Nishioka, K.: A Proposal for a Japanese Keyboard on Cellular Phones. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 299–306. Springer, Heidelberg (2008)
34. Margenstern, M., Morita, K.: NP problems are tractable in the space of cellular automata in the hyperbolic plane. Theoretical Computer Science 259, 99–128 (2001)
35. Margenstern, M., Skordev, G.: Fibonacci Type Coding for the Regular Rectangular Tilings of the Hyperbolic Plane. Journal of Universal Computer Science 9(5), 398–422 (2003)
36. Margenstern, M., Skordev, G.: Tools for devising cellular automata in the hyperbolic 3D space. Fundamenta Informaticae 58(2), 369–398 (2003)
37. Margenstern, M., Song, Y.: A universal cellular automaton on the ternary heptagrid. Electronic Notes in Theoretical Computer Science 223, 167–185 (2008)
38. Margenstern, M., Song, Y.: A new universal cellular automaton on the pentagrid. Parallel Processing Letters 19(2), 227–246 (2009)
39. Martin, B.: VirHKey: a VIRtual Hyperbolic KEYboard with gesture interaction and visual feedback for mobile devices. In: Mobile HCI 2005, pp. 99–106 (2005)
40. Morgenstein, D., Kreinovich, V.: Which Algorithms are Feasible and Which are not Depends on the Geometry of Space-Time. Geocombinatorics 4(3), 80–97 (1995)
41. Moore, E.F.: Machine Models of Self-reproduction. In: Proceedings of the Symposium in Applied Mathematics, vol. 14, pp. 17–33 (1962)
42. Myhill, J.: The Converse to Moore's Garden-of-Eden Theorem. In: Proceedings of the American Mathematical Society, vol. 14, pp. 685–686 (1963)
43. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. Inventiones Mathematicae 12, 177–209 (1971)
44. Robinson, R.M.: Undecidable tiling problems in the hyperbolic plane. Inventiones Mathematicae 44, 259–264 (1978)
45. Stewart, I.: A Subway Named Turing, Mathematical Recreations in Scientific American, pp. 90–92 (1994)

Flattening the Transition P Systems with Dissolution

Oana Agrigoroaiei and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science
Blvd. Carol I no.8, 700505 Iaşi, Romania
oanaag@iit.tuiasi.ro
"A.I.Cuza" University, Blvd. Carol I no.11, 700506 Iaşi, Romania
gabriel@info.uaic.ro

Abstract. Given a transition P system Π with dissolution, promoters and inhibitors having several membranes, we construct a P system Π^f with promoters and inhibitors and with only one membrane. The evolution of this "flat" P system Π^f simulates the evolution of initial transition P system Π by replacing any dissolution stage of a configuration in Π by specific rules application in a configuration of Π^f. The transition P systems without dissolution represent a special case.

1 Introduction

Membrane systems (also called P systems) represent a biologically inspired model of computation, involving parallel application of rules, communication between membranes and membrane dissolution. Membrane systems are introduced by Gh.Păun and presented in monograph [5] as a class of distributed parallel computing devices inspired by biology. This computing model is represented by complex hierarchical structures with a flow of materials and information which supports their functioning. Essentially, the membrane systems are composed of various compartments with different tasks, all of them working simultaneously to accomplish a more general task.

The motivation of constructing a P system with only one membrane which simulates a P system with multiple membranes and dissolution is to use the former in place of the latter without reducing the generality of a problem. In the conclusion of this paper we discuss the constraints of such a replacement.

We construct the "flat" P system Π^f by replacing objects in membranes of Π with pairs of objects and labels of membranes. Each rule r of Π is translated into sets of rules r^f for Π^f, and dissolution of a membrane labelled by i in Π is translated into the use of rules from a set D_i for Π_f. An evolution step of Π is translated into a single evolution step of Π^f whenever the rules applied in Π do not involve dissolution, and into two evolution steps in Π^f whenever they do. In the second case the first step of Π^f corresponds to rule application in Π and uses only rules from the sets r^f, while the second step corresponds to dissolution of membranes in Π and uses only rules from the sets D_i together with a special rule $\nabla \to 0$ which acts like a semaphore for dissolution.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 53–64, 2010.

1.1 P Systems and Multisets

We assume the reader is familiar with membrane computing [5]. For readability, here we present only the necessary notions.

A *P system* of degree m is a tuple $\Pi = (O, \mu, R_1, \ldots, R_m)$, where

- O is an alphabet of objects;
- μ is a membrane structure, with the membranes labelled by natural numbers $1 \ldots m$, in a one-to-one manner;
- each R_i is a finite set of rules associated with the membrane labelled by i; the rules have the form $u \to v$, where u is a non-empty multiset of objects and v a multiset over objects a and messages of the form $(a, out), (a, in_j), \delta$ with the condition that δ can appear at most once;

The rules of the *skin* membrane (which is labelled by 1) do not involve the special symbol δ. This symbol, whenever it is produced by a rule, marks the dissolution of the membrane in which it appears. By dissolution we understand that, after applying rules in a maximally parallel manner, all the objects of a membrane in which δ is produced are sent to its parent membrane. The membrane itself no longer exists, thus modifying the structure of the system.

We can associate *promoters* and *inhibitors* with a rule $u \to v$, in the form $(u \to v)|_{prom, \neg inhib}$, with $prom, inhib$ sets of objects from O. Such a rule associated with a membrane i is applied to a multiset w of objects only if every element of $prom$ is present in w ($w(a) \geq 1, \forall a \in prom$) and no element of $inhib$ is present in w ($w(a) = 0, \forall a \in inhib$). If one or both of the sets $prom, inhib$ is empty, we write the rule $(u \to v)|_{prom, \neg inhib}$ as $(u \to v)|_{prom}$ or $(u \to v)|_{\neg inhib}$ or simply $u \to v$. The promoters and inhibitors of membrane systems formalize the reaction enhancing and reaction prohibiting roles of various substances present in cells [3]. Several papers use rules having at most one object as promoter and one as inhibitor; here we consider sets of promoters and inhibitors. Other generalizations are common in the literature (for instance [1]).

We consider a multiset w over a set S to be a function $w : S \to \mathbb{N}$. When describing a multiset characterized for instance, by $w(s) = 1, w(t) = 2, w(s') = 0, s' \in S \backslash \{s, t\}$, we use the representation $s + 2t$. To each multiset w we associate its support, denoted by $supp(w)$, which contains those elements of S which have a non-zero image. A multiset is called non-empty if it has non-empty support. We denote the empty multiset by 0_S or by 0 when the set over which the multiset is defined is clear from the context. We overload the set notation to multisets by using $s \in w$ instead of $w(s) \geq 1$.

The sum of two multisets w, w' over S is the multiset $w + w' : S \to \mathbb{N}, (w + w')(s) = w(s) + w'(s)$. For two multisets w, w' over S we say that w is contained in w' if $w(s) \leq w'(s), \forall s \in S$. We denote this by $w \leq w'$. If $w \leq w'$ we can define $w' - w$ by $(w' - w)(s) = w'(s) - w(s)$.

2 A Simple Semantics of P Systems

We call configuration of a P system a vector of length m (which is the degree of the P system) whose elements are each either a multiset over O or a

special symbol $*$. We denote by $\mathcal{C}_{\mathbf{T}}(\Pi)$ the set of configurations for Π. We call intermediate configuration a vector of length m whose elements are either a multiset over $O \cup \{\delta\}$ or the special symbol $*$. We denote by $\mathcal{C}^{\#}_{\mathbf{T}}(\Pi)$ the set of intermediate configurations for Π. Note that $\mathcal{C}_{\mathbf{T}}(\Pi) \subseteq \mathcal{C}^{\#}_{\mathbf{T}}(\Pi)$.

Before we proceed, we introduce some new notations and definitions.

For $W = (w_1, \ldots, w_m) \in \mathcal{C}^{\#}_{\mathbf{T}}(\Pi)$ we denote by $\Delta(W)$ the set of labels $\{i \in \{1, \ldots, m\} / w_i = *\}$ of previously dissolved membranes.

We use $\mu(i) = \mu^1(i)$ to denote the parent of the membrane i with respect to the membrane structure μ. We use $\mu^{k+1}(i)$ to denote the parent of the membrane $\mu^k(i)$. We let μ_W denote the map giving the membrane structure with respect to $W \in \mathcal{C}^{\#}_{\mathbf{T}}(\Pi)$, defined by

$$\mu_W(j) = \begin{cases} \text{undefined, if } j \in \Delta(W) \\ \mu(j), \text{ if } j, \mu(j) \notin \Delta(W) \\ \mu^{l+1}(j), \text{ if } j \notin \Delta(W), \mu(j), \ldots, \mu^l(j) \in \Delta(W), \mu^{l+1}(j) \notin \Delta(W). \end{cases}$$

In other words, $\mu_W(j)$ discards the dissolved membranes which are candidates for parents of j until it reaches an undissolved one and chooses it as the current parent of j.

Definition 1. *We consider a label i and a configuration $W = (w_1, \ldots, w_m)$. A family of multisets $\mathcal{R}_i$ over R_i is called* valid *with respect to W whenever $lhs(\mathcal{R}_i) \leq w_i$ and for each rule $r : (u \to v)|_{prom, \neg inhib}$ such that $r \in \mathcal{R}_i$ we have: (i) $w_i(a) \geq 1, \forall a \in prom$; (ii) $w_i(a) = 0, \forall a \in inhib$; and (iii) if $(a, in_j) \in v$ then $\mu_W(j) = i$.*

A family of multisets of rules $\{\mathcal{R}_j\}_{j \in \{1,\ldots,m\}}$, with each $\mathcal{R}_j$ over R_j, is said to be maximally valid *with respect to a configuration W if it is valid with respect to W and moreover it is maximal with this property: if another family of multisets of rules $\mathcal{R}'_j$ is valid with respect to W and $\mathcal{R}_j \subseteq \mathcal{R}'_j$ then $\mathcal{R}_j = \mathcal{R}'_j$, for all $j \in \{1, \ldots, m\}$.*

We remark that there can exist rules r in R_i producing objects of form (a, in_j) where j is not a child of i in the initial membrane structure μ. Such rules can become active after successive dissolutions.

We define the maximal rewriting stage, which in this semantics includes message sending.

Definition 2. *For $W = (w_1, \ldots, w_m) \in \mathcal{C}_{\mathbf{T}}(\Pi)$ and $V = (v_1, \ldots, v_m) \in \mathcal{C}^{\#}_{\mathbf{T}}(\Pi)$ we define $(w_1, \ldots, w_m) \to_{mpr} (v_1, \ldots, v_m)$ if and only if the following hold:*

- *for all $i \in \{1, \ldots, m\} \backslash \Delta(W)$ there exist multisets of rules $\mathcal{R}_i$ over R_i which are maximally valid with respect to W and such that for each $r \in \mathcal{R}_i$ and $(a, in_j) \in rhs(r)$ we have $j \notin \Delta(W)$;*
- *for all $i \in \{1, \ldots, m\} \backslash \Delta(W)$, v_i is given by $v_i(\delta) = 1$ if $rhs(\mathcal{R}_i)(\delta) > 1$ and*

$$v_i(a) = w_i(a) - lhs(\mathcal{R}_i)(a) + rhs(\mathcal{R}_i)(a) + \\ + rhs(\mathcal{R}_{\mu_W(i)})(a, in_i) + \sum_{j \in \mu_W^{-1}(i)} rhs(\mathcal{R}_j)(a, out) \qquad (1)$$

– *for all* $i \in \Delta(W)$, $v_i = *$.

We now define the dissolution stage.

Definition 3. *For* $(u_1, \ldots, u_m) \in \mathcal{C}_{\mathbf{T}}^{\#}(\Pi)$ *and* $(v_1, \ldots, v_m) \in \mathcal{C}_{\mathbf{T}}(\Pi)$ *we define* $(u_1, \ldots, u_m) \rightarrow_\delta (v_1, \ldots, v_m)$ *if and only if the following hold:*

– *there is at least one label* i *such that* $u_i(\delta) = 1$*;*
– *let* $U' = (u'_1, \ldots, u'_m) \in \mathcal{C}_{\mathbf{T}}^{\#}(\Pi)$ *be given by* $u'_i = *$ *if* $u_i = *$ *or* $u_i(\delta) = 1$ *and* $u'_i = 0$ *otherwise;*
– $v_i = *$ *if* $u'_i = *$ *and* $v_i = u_i + \sum\{u_j - \delta \mid u'_i \neq *, u_j(\delta) = 1, \mu_{U'}(j) = i\}$ *otherwise.*

We ask that there exists at least one label i such that $u_i(\delta) = 1$ since otherwise there is no need for a dissolution step. We construct U' to provide a "skeleton" for the new membrane structure (note that $\Delta(U) = \Delta(U')$) so we know where to send the contents of the membranes we just dissolved (those j with $u_j(\delta) = 1$), namely to $i = \mu_{U'}(j)$.

We define the transition system $\mathbf{T}$ on $\mathcal{C}_{\mathbf{T}}(\Pi)$ by setting

$$W \Longrightarrow_{\mathbf{T}} V \textbf{ if and only if } W \rightarrow_{mpr} V \text{ or } W \rightarrow_{mpr} \rightarrow_\delta V$$

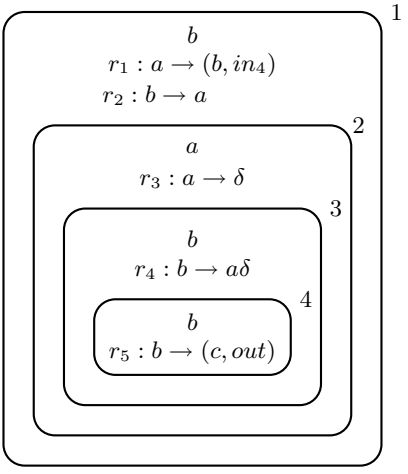


Fig. 1. Membrane structure, rules and the configuration W

Example 1. We use a running example of a P system with 4 membranes, with $\mu(4) = 3, \mu(3) = 2, \mu(2) = 1$, with $R_1 = \{r_1 : a \rightarrow (b, in_4), r_2 : b \rightarrow a\}$, $R_2 = \{r_3 : a \rightarrow \delta\}$, $R_3 = \{r_4 : b \rightarrow a\delta\}$ and $R_4 = \{r_5 : b \rightarrow (c, out)$. Consider a configuration $W = (b, a, b, b)$ (see Figure 1).

The evolution of the system, starting from configuration W, is:

$$W \rightarrow_{mpr} (a, \delta, a + c + \delta, 0) \rightarrow_\delta (2a + c, *, *, 0) \rightarrow_{mpr} (c, *, *, 2b) \rightarrow_{mpr} (3c, *, *, 0)$$

3 Flattening Membrane Systems with Dissolution

Consider a transition P system Π with dissolution; in order to simplify our presentation, we assume its rules do not involve promoters and inhibitors. However the procedure can be applied to transition P systems with promoters and inhibitors, by adding the promoters and inhibitors of each rule r to the "flat" rules corresponding to r. Starting from a system Π, we construct a P system Π^f with only one membrane and with rules involving promoters and inhibitors such that the evolution of Π corresponds to the evolution of Π^f. In more detail, a maximal parallel rewriting stage in Π corresponds to one in Π^f, and a dissolution stage in Π corresponds to a maximal parallel rewriting stage in Π^f.

Definition 4. *We say that $i \in \{1, \ldots, m\}$ is* dissoluble *if there exists a rule $r \in R_i$ such that $\delta \in rhs(r)$.*

If a membrane is not dissoluble then it will never be dissolved in any possible evolution of any initial configuration. Thus the skin membrane is not dissoluble. On the other hand, if a membrane is dissoluble, then it might be dissolved depending on the initial configuration chosen and on its evolution.

Definition 5. *We define by $maxparent(i)$ the "most remote" membrane which can become a parent of i, after possible dissolutions of membranes. In more detail, $maxparent(i)$ is defined by*

$$maxparent(i) = \mu^k(i) \text{ where } k = \min\{n \geq 1 \mid \mu^n(i) \text{ is not dissoluble}\}.$$

We let $l(i)$ denote k such that $\mu^k(i) = maxparent(i)$.

In **Example 1**, $maxparent(4) = maxparent(3) = maxparent(2) = 1$ and $l(4) = 3, l(3) = 2, l(2) = 1$. The configuration W is chosen such that membrane 4 does indeed become a child of the skin membrane after one evolution step.

For a multiset w over $O \cup O \times \{in_1, \ldots, in_n\} \cup \{\delta, *\}$ we denote by (w, i) the multiset over $(O \cup \{\delta\}) \times \{1, \ldots, m\}$ obtained by adding to every object $a \in O \cup \{\delta\}$ the label i, replacing in_j by j and replacing $*$ by (δ, i):

- $(w, i)(a, i) = w(a) + w(a, in_i)$;
- $(w, i)(a, j) = w(a, in_j)$, for $j \neq i$;
- $(w, i)(\delta, i) = w(\delta) + w(*)$;
- $(w, i)(\delta, j) = 0$ for $j \neq i$.

Note that by $(w, i)(a, i)$ we understand function application, not string concatenation, since we do not use a string representation for multisets.

The P system Π^f is defined by $\Pi^f = (O^f, \mu^f, R^f)$ with components defined as follows. The alphabet O^f of Π^f is $(O \cup \{\delta\}) \times \{1, \ldots, m\} \cup \{\nabla\}$. The objects (δ, i) are used to represent the fact that the membrane labelled by i is dissolved or undergoing dissolution in Π. The special symbol ∇ is used to separate between the application of rules in Π^f which correspond to rules in Π and the application of rules in Π^f which simulate dissolution in Π. The membrane structure μ^f

contains only one membrane. The set R^f of rules consists of a special rule $\nabla \to 0$ together with the union of sets of rules r^f for each rule r of the P system Π and the union of sets of rules D_i for each dissoluble i. Formally,

$$R^f = \bigcup_{\substack{r \in R_i \\ i \in \{1,\dots,m\}}} r^f \quad \cup \quad \{\nabla \to 0\} \quad \cup \bigcup_{\substack{i \in \{1,\dots,m\} \\ i \text{ dissolvable}}} D_i$$

Note that the right hand side of the rule $\nabla \to 0$ is the empty multiset 0, meaning that no objects are produced by its application.

We define now the set r_f of rules in Π^f to simulate the application of the corresponding rule r of Π, and the set D_i of rules of Π^f to simulate the dissolution of membrane i in Π. Namely, these rule sets are defined in a manner which ensures that in each evolution step of the "flat" system Π^f we either apply rules from the sets r^f or rules from the sets D_i together with the special rule $\nabla \to 0$.

Definition 6. *For each rule r of Π we define the corresponding set r^f of rules in Π^f. We start by defining $prom(r) = \{(\delta, \mu^l(j)) \mid \exists (a, in_j) \in rhs(r) : \mu^k(j) = i, 0 < l < k\}$.*

1. *For each rule $u \to v \in R_i$ such that v contains no out messages and $v(\delta) = 0$, r^f contains only the rule $\underline{r} : (u, i) \to (v, i)|_{prom(r), \neg\{\nabla\}}$;*
2. *for each rule $u \to v \in R_i$ such that v contains no out messages and $v(\delta) = 1$, r^f contains only the rule $\underline{r} : (u, i) \to (v, i)\nabla|_{prom(r), \neg\{\nabla\}}$;*
3. *for each rule $u \to (v, out)w \in R_i$ such that w contains no out messages and $w(\delta) = 0$, r^f is the following set of rules*

$$r^f = \{\underline{r}_k : (u, i) \to (v, \mu^k(i))(w, i)|_{prom_k(i), \neg inh_k(i)} \mid k \in \{1, \dots, l(i)\}\},$$

where the sets of promoters $prom_k(i)$ and inhibitors $inh_k(i)$ are defined by
   - *$prom_1(i) = prom(r)$, $inh_1(i) = \{\nabla, (\delta, \mu(i))\}$;*
   - *$prom_2(i) = prom(r) \cup \{(\delta, \mu(i))\}$, $inh_1(i) = \{\nabla, (\delta, \mu^2(i))\}$;*
   - *...*
   - *$prom_{l(i)-1}(i) = prom(r) \cup \{(\delta, \mu(i)), \dots (\delta, \mu^{l(i)-2}(i))\}$, $inh_{l(i)-1}(i) = \{\nabla, (\delta, \mu^{l(i)-1}(i))\}$;*
   - *$prom_{l(i)}(i) = prom(r) \cup \{(\delta, \mu(i)), \dots (\delta, \mu^{l(i)-1}(i))\}$, $inh_{l(i)}(i) = \{\nabla\}$;*
4. *for each rule $u \to (v, out)w \in R_i$ such that w contains no out messages and $w(\delta) = 1$, r^f is the following set of rules*

$$r^f = \{\underline{r}_k : (u, i) \to (v, \mu^k(i))(w, i)\nabla|_{prom_k(i), \neg inh_k(i)} \mid k \in \{1, \dots, l(i)\}\},$$

where $prom_k(i)$ and $inh_k(i)$ are defined as for the previous type of rule.

In **Example 1**, the sets of rules r^f are as follows:

- from rule $r_1 : a \to (b, in_4)$ we get $r_1^f = \{(a, 1) \to (b, 4)|_{\{(\delta,2),(\delta,3)\}, \neg\{\nabla\}}\}$;
- from rule $r_2 : b \to a$ we get $r_2^f = \{(b, 1) \to (a, 1)|_{\neg\{\nabla\}}\}$;
- from rule $r_3 : a \to \delta$ we get $r_3^f = \{(a, 2) \to (\delta, 2)|_{\neg\{\nabla\}}\}$;

- from rule $r_4 : b \to a\delta$ we get $r_4^f = \{(b,3) \to (a+\delta,3)|_{\neg\{\nabla\}}\}$
- from rule $r_5 : b \to (c, out)$ we get r_5^f containing the following rules:
  - $(b,4) \to (c,3)|_{\neg\{\nabla,(\delta,3)\}}$;
  - $(b,4) \to (c,2)|_{\{(\delta,3)\},\neg\{\nabla,(\delta,2)\}}$;
  - $(b,4) \to (c,1)|_{\{(\delta,3),(\delta,2)\},\neg\{\nabla\}}$.

The set $prom(r)$ is used to ensure that a rule from r^f can only be applied if, whenever a message (a, in_j) appears in the right hand side of r, the membrane with label j is the current child of membrane i after several dissolutions. This means that if other membranes $\mu^l(j)$ existed between j and $i = \mu^k(j)$, they should be dissolved before r can be applied. Translated to Π^f and r^f, this becomes a requirement for $(\delta, \mu^l(j))$ to be present before the rules from r^f can be applied.

When the right hand side of the rule r does not contain *out* messages, the choice for r^f is straightforward: flatten the multisets in the right and left hand side of r, and if r involves dissolution, we add a special symbol ∇ to the left hand side. The symbol ∇ is also added as an inhibitor to ensure that until ∇ is consumed these rules are not applied. The idea is that ∇ can only be consumed in the stage simulating the dissolution of membranes in Π, which follows whenever ∇ is produced.

When the right hand side of the rule r does contain *out* messages, the rules of the set r^f are defined to ensure that the destination of the messages is the first undissolved parent of i. For example, for $k = 1$, we have in r^f the rule

$$\underline{r}_1 : (u,i) \to (v,\mu(i))(w,i)|_{\neg\{\nabla,(\delta,\mu(i))\}}$$

which "replaces" (v, out) with $(v, \mu(i))$ and can be applied only when $\mu(i)$ is not dissolved (and ∇ is not present). For $k = 2$, we have in r^f a rule

$$\underline{r}_2 : (u,i) \to (v,\mu^2(i))(w,i)|_{\{(\delta,\mu(i))\},\neg\{\nabla,(\delta,\mu^2(i))\}}$$

which can be applied only when $\mu(i)$ is dissolved and $\mu^2(i)$ is not dissolved (and ∇ is not present). Note that $inh_{l(i)}$ contains only ∇ because by definition $\mu^{l(i)}(i) = maxparent(i)$ cannot be dissolved.

Whenever we consider P systems with promoters and inhibitors, we should add the set of promoters and the set of inhibitors of rule r to the set of promoters and the set of inhibitors of each rule in r^f, respectively.

Definition 7. *For each dissoluble label i we define the corresponding set D_i of rules in Π^f as follows:*

$$D_i = \{d_{a,i,k} : (a,i) \to (a,\mu^k(i))|_{prom'_k(i),\neg inh'_k(i)} \mid a \in O, k \in \{1,\ldots,l(i)\}\},$$

where the sets of promoters $prom'_k(i)$ and inhibitors $inh'_k(i)$ are defined by

- $prom'_1(i) = \{\nabla, (\delta,i)\}$, $inh'_1(i) = \{(\delta,\mu(i))\}$;
- $prom'_2(i) = \{\nabla, (\delta,i), (\delta,\mu(i))\}$, $inh'_1(i) = \{(\delta,\mu^2(i))\}$;
- ...

- $prom'_{l(i)-1}(i) = \{\nabla, (\delta, i), \ldots (\delta, \mu^{l(i)-2}(i)\}$, $inh'_{l(i)-1}(i) = \{(\delta, \mu^{l(i)-1}(i))\}$;
- $prom'_{l(i)}(i) = \{\nabla, (\delta, i), \ldots (\delta, \mu^{l(i)-1}(i)\}$, $inh'_{l(i)}(i) = \emptyset$.

The object (δ, i) stands for the fact that the membrane labelled by i is dissolved or undergoing dissolution in Π. If membrane i is dissolved in Π, no objects (a, i) exist in Π^f, and so no rules from D_i can apply. If it undergoes dissolution, then one of the rules in D_i will be applied such that all (a, i) are transformed into objects (a, j), where j is the first undissolved parent of i. The choice for j is made using promoters and inhibitors in a similar manner to the replacement of the message *out* in the sets r^f. The object ∇ is used as promoter to ensure that the rules from D_i are applied only to simulate dissolution in Π, namely, after some object (δ, i) is produced (together with ∇) by some rule from one of the sets r^f of Π^f.

In **Example 1**, the sets D_i are defined for $i \in \{2, 3\}$ and $O = \{a, b, c\}$. The set D_2 contains rules of form $(x, 2) \to (x, 1)|_{\{\nabla,(\delta,2)\}}$ for $x \in O$. The set D_3 contains rules of form $(x, 3) \to (x, 2)|_{\{\nabla,(\delta,3)\},\neg\{(\delta,2)\}}$ or of form $(x, 3) \to (x, 1)|_{\{\nabla,(\delta,3),(\delta,2)\}}$ for $x \in O$.

We summarise the general ideas behind the construction of Π^f in Figure 2.

Π **with** m **membranes**	Π^f **with** 1 **membrane**
object a in membrane i	object (a, i)
δ appears in membrane i	object (δ, i)
rule r without "out"	a rule r^f
rule r with "out"	set r^f of rules (possible parents)
membrane i dissoluble	set D_i of rules $(x, i) \to (x, cPar_i)$
mpr step in Π (including communication)	*mpr* step in Π^f using r^f
diss step in Π	*mpr* step in Π^f using D_i
–	special rule $\nabla \to 0$ to ensure separation between applying rules from r^f and rules from D_i

Fig. 2. Correspondence between Π and Π^f

For an intermediate configuration $W = (w_1, \ldots, w_n)$ of Π let $flat(W)$ denote the configuration of Π^f defined by

- $flat(W)(a, i) = w_i(a)$;
- $flat(W)(\delta, i) = 1$ if $w_i(\delta) = 1$ or $w_i = *$; $flat(W)(\delta, i) = 0$ otherwise;
- $flat(W)(\nabla) = \sum_i w_i(\delta)$.

Note that if W is a configuration, $flat(W)$ does not contain the special symbol ∇.

Theorem 1. *For two configurations W and V of Π,*

if $W \Longrightarrow_{\mathbf{T}} V$ **then** $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$ *or* $flat(W) \Longrightarrow_{\mathbf{T}} \Longrightarrow_{\mathbf{T}} flat(V)$;
if $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$ **then** $W \Longrightarrow_{\mathbf{T}} V$.

Proof. Let $W = (w_1, \ldots, w_m)$ and $V = (v_1, \ldots, v_m)$.
First Implication: Firstly, suppose that the transition $W \Longrightarrow_{\mathbf{T}} V$ does not involve dissolution, in other words $W \rightarrow_{mpr} V$. We prove that in this case $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$.

Since $W \rightarrow_{mpr} V$, for each $i \in \{1, \ldots, m\}$ there exists a multiset $\mathcal{R}_i$ of rules from R_i such that the family of multisets of rules is maximally valid with respect to W. Moreover, $\Delta(V) = \Delta(W)$ and $lhs(r)(\delta) = 0, \forall r \in \mathcal{R}_i$. We also know that for each $i \in \{1, \ldots, m\} \backslash \Delta(W)$ v_i are obtained from w_i as in Equation (1).

Let $\mathcal{R}$ be the multiset of rules from R^f defined by

- $\mathcal{R}(s) = \mathcal{R}_i(r)$ if $s = \underline{r}$, $r \in R_i$;
- $\mathcal{R}(s) = \mathcal{R}_i(r)$ if $s = \underline{r}_k$, $r \in R_i$ and $\mu^k(i) = \mu_W(i)$;
- $\mathcal{R}(s) = 0$ if $s = \underline{r}_k$, $r \in R_i$ and $\mu^k(i) \neq \mu_W(i)$;

with $\mathcal{R}(s) = 0$ if s is the special rule $\nabla \rightarrow 0$ or if s belongs to one of the sets D_i. We prove that $\mathcal{R}$ is maximally valid in $flat(W)$ and that $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$ by using $\mathcal{R}$.

Note that $lhs(\mathcal{R})(a, i) = (lhs(\mathcal{R}_i), i)$ because when defining $\mathcal{R}$ we always take only one rule from each r^f with the multiplicity that r has in $\mathcal{R}_i$. Hence $\mathcal{R}$ is valid in $flat(W)$. If it was not maximally valid, there would be a rule $s \in R^f$ such that $\mathcal{R} + s$ is valid in $flat(W)$. Since $flat(W)$ does not contain ∇, s cannot be $\nabla \rightarrow 0$; nor can it be a rule of form $d_{a,i,k}$ because if $flat(W)$ contains the promoter (δ, i) of $d_{a,i,k}$ then $flat(W)(a, i) = 0, \forall a \in O$. So the rule s would be either $\underline{r}$ or $\underline{r}_k$ for some j and some rule $r \in R_j$. This, however, contradicts the maximal validity of the family $\mathcal{R}_i$ since we would have $lhs(\mathcal{R}_j + r) \leq w_j$.

To prove $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$ we prove that $flat(V) = flat(W) - lhs(\mathcal{R}) + rhs(\mathcal{R})$. We check the identity for all $(x, i) \in (O \cup \{\delta\}) \times \{1, \ldots, m\}$. For all $i \in \Delta(W)$ we have $flat(V)(x, i) = flat(W)(x, i)$ and $lhs(\mathcal{R})(x, i) = rhs(\mathcal{R})(x, i) = 0$. For all $i \in \{1, \ldots, m\} \backslash \Delta(W)$ the identity is inferred from

$$rhs(\mathcal{R})(a, i) = rhs(\mathcal{R}_i)(a) + rhs(\mathcal{R}_{\mu_W(i)})(a, in_i) + \sum_{j \in \mu_W^{-1}(i)} rhs(\mathcal{R}_j)(a, out) \quad (2)$$

which is proved by

$$rhs(\mathcal{R})(a, i) = \sum_j \sum_{r \in \mathcal{R}_j} \sum_{s \in r^f} \mathcal{R}(s) \cdot rhs(s)(a, i) =$$

$$\sum_{j=i} \sum_{r \in \mathcal{R}_j} \mathcal{R}_j(r) \cdot rhs(r)(a, i) + \sum_{j=\mu_W(i)} \sum_{r \in \mathcal{R}_j} \mathcal{R}_j(r) \cdot rhs(r)(a, in_i) +$$

$$+ \sum_{\mu_W(j)=i} \sum_{r \in \mathcal{R}_j} \mathcal{R}_j(r) \cdot rhs(r)(a, out).$$

Secondly, suppose that the transition $W \Longrightarrow_{\mathbf{T}} V$ does involve dissolution, in other words $W \rightarrow_{mpr} \rightarrow_{\delta} V$. Then there exists an intermediate configuration

$U = (u_1, \ldots, u_m)$ and multisets of rules $\mathcal{R}_i, \forall i \in \{1, \ldots, m\}$ such that $W \rightarrow_{mpr} U$ by using $\mathcal{R}_i$ and $U \rightarrow_\delta V$. Let $flat(U)$ be the configuration of Π^f corresponding to U.

The proof that $flat(W) \Longrightarrow_{\mathbf{T}} flat(U)$ is similar to the proof that $W \Longrightarrow_{\mathbf{T}} V$ implies $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$. Moreover, we note that $flat(U)(\nabla) > 0$. We prove now that $flat(U) \Longrightarrow_{\mathbf{T}} flat(V)$. Let $\mathcal{D}$ be the multiset of rules over R^f defined by $\mathcal{D}(s) = 0, \forall s \in r^f$, $\mathcal{D}(\nabla \rightarrow 0) = flat(U)(\nabla)$ and

- $\mathcal{D}(d_{a,i,k}) = 0$ if $flat(U)(\delta, i) = 0$ or $\mu^k(i) \neq \mu_V(i)$;
- $\mathcal{D}(d_{a,i,k}) = flat(U)(a, i)$ if $flat(U)(\delta, i) > 0$ and $\mu^k(i) = \mu_V(i)$.

We prove that $\mathcal{D}$ is maximally valid in $flat(U)$ and that $flat(U)$ evolves to $flat(V)$ using $\mathcal{D}$. The validity of $\mathcal{D}$ follows from its definition. To see that it is also maximal with this property, suppose there exists $s \in R^f$ such that $\mathcal{D} + s$ is valid in $flat(U)$. Clearly s cannot be a rule from one of the sets r^f since all those rules are inhibited by ∇. Also, s cannot be the special rule $\nabla \rightarrow 0$ because $\mathcal{D}(\nabla \rightarrow 0) = flat(U)(\nabla)$. Suppose s is one of the rules $d_{a,i,k}$. If $flat(U)(\delta, i) > 0$ this contradicts $\mathcal{D}(d_{a,i,k}) = flat(U)(a, i)$. If $flat(U)(\delta, i) = 0$ then the promoter (δ, i) of $s = d_{a,i,k}$ is missing from $flat(U)$.

To prove that $flat(U)$ evolves to $flat(V)$ using $\mathcal{D}$, we show that $flat(V) = flat(U) - lhs(\mathcal{D}) + rhs(\mathcal{D})$. To this purpose, note that $flat(V)(a, i) = 0 = flat(U)(a, i) - lhs(\mathcal{D})(a, i)$ and $rhs(\mathcal{D})(a, i) = 0$ for all $a \in O$ and i such that $(\delta, i) \in flat(U)$. Moreover, for those i such that $flat(U)(\delta, i) = 0$ we have $flat(V)(a, i) = flat(U)(a, i) + rhs(\mathcal{D})(a, i)$ and $lhs(\mathcal{D})(a, i) = 0$, $\forall a \in O$.

Second Implication: First suppose that $flat(W) \Longrightarrow_{\mathbf{T}} flat(V)$ by using a multiset $\mathcal{R}$ of rules over R^f. Since $flat(W)$ does not contain ∇, the multiset $\mathcal{R}$ of rules cannot contain rules from the sets D_i nor the special rule $\nabla \rightarrow 0$. Moreover, it cannot contain rules which have (δ, i) in the right hand side (because (δ, i) is always accompanied by ∇). From the way the sets r^f are defined, a valid multiset of rules $\mathcal{R}$ cannot contain two distinct rules from the same set r^f. Thus we can define the multisets $\mathcal{R}_i$ over each set R_i in Π by $\mathcal{R}_i(r) = \mathcal{R}(s)$ if there exists $s \in r^f$ such that $\mathcal{R}(s) > 0$ (if it exists, s is unique) and $\mathcal{R}_i(r) = 0$ otherwise.

We prove that the family of multisets $\mathcal{R}_i$ of rules is maximally valid with respect to W and that V is obtained from W by using it. For validity, it suffices to see that $lhs(\mathcal{R}_i) \leq w_i$ and that if $(a, in_j) \in rhs(r)$ then we obtain that $\mu_W(j) = i$, because $flat(W)(\delta, \mu^s(j)) = 1$ for all $(\delta, \mu^s(j)) \in prom(r)$. For maximal validity, suppose there exists some family of multisets $\mathcal{R}'_i$ valid with respect to W such that $\mathcal{R}_i \leq \mathcal{R}'_i$. Then the multiset $\mathcal{R}'$ of rules defined as in the proof of the first implication is valid with respect to $flat(W)$ and $\mathcal{R} \leq \mathcal{R}'$ which implies that $\mathcal{R}_i = \mathcal{R}'_i$. To see that V is obtained from W by using $\mathcal{R}_i$ we just use identity (2) which holds for the multisets $\mathcal{R}$ and $\mathcal{R}_i$ defined above.

Proposition 1. *For two configurations W and V of Π and a configuration X of Π^f such that $\nabla \in X$,* **if** $flat(W) \Longrightarrow_{\mathbf{T}} X \Longrightarrow_{\mathbf{T}} flat(V)$ **then** $W \Longrightarrow_{\mathbf{T}} V$.

Proof. Let $W = (w_1, \ldots, w_m)$ and $V = (v_1, \ldots, v_m)$.

Let $\mathcal{R}$ be the multiset of rules used in $flat(W) \Longrightarrow_{\mathbf{T}} X$. Since $\nabla \notin flat(W)$ it follows (exactly as in the proof of the second implication of Theorem 1) that $W \rightarrow_{mpr} U$ for a intermediate configuration U. Moreover, we obtain that $flat(U) = X$.

Let $\mathcal{D}$ be the multiset of rules used in $X \Longrightarrow_{\mathbf{T}} flat(V)$. Since $\nabla \in X$, any rule in $\mathcal{D}$ is either from the sets D_i or the special rule $\nabla \rightarrow 0$. Clearly, $\mathcal{D}(\nabla \rightarrow 0) = X(\nabla)$. Moreover, for each i such that $(\delta, i) \in X$ and for each a such that $(a, i) \in X$, there can be exactly one $k = k(i) \leq l(i)$ such that $\mathcal{D}(d_{a,i,k}) > 0$. If either (δ, i) or (a, i) does not appear in X, then $\mathcal{D}(d_{a,i,l}) = 0$ for all $l \leq l(i)$.

We prove that $U \rightarrow_{\delta} V$. Consider $U' = (u'_1, \ldots, u'_m)$ as in Definition 3. Then for $k(i)$ previously defined we have $\mu^{k(i)}(i) = \mu_{U'}(i)$ (because $X = flat(U)$). Thus for all $j \notin \Delta(V)$ we have

$$v_j(a) = flat(V)(a, j) = X(a, j) + \sum_i \{X(a, i) \mid (\delta, i) \in X, \mu_{U'}(i) = j\}$$

which implies $v_i = u_i + \sum\{u_j - \delta \mid u'_i \neq *, u_j(\delta) = 1, \mu_{U'}(j) = i\}$, and so concluding the proof.

For P systems without dissolution, Theorem 1 and Proposition 1 can be combined into a single result.

Corollary 1. *Let Π be a P system without dissolution and Π^f its associated P system with only one membrane. For W and V configurations of Π,*

$$W \Longrightarrow_{\mathbf{T}} V \textbf{ if and only if } flat(W) \Longrightarrow_{\mathbf{T}} flat(V).$$

Remark 1. We end by emphasizing the size of the P system Π^f with respect to that of Π. Thus, the cardinality of the alphabet O^f is $card(O^f) = (card(O) + 1) \cdot m + 1$, while the cardinality of the rule set R^f is, according to Definitions 6 and 7,

$$card(R^f) = \sum_{i \in \{1,\ldots,m\}} \sum_{r \in R_i} (card\{r \in R_i \mid \nexists (a, out) \in rhs(r)\} +$$
$$+ l(i) \cdot card\{r \in R_i \mid \exists (a, out) \in rhs(r)\}) + \sum_{i \text{ dissolvable}} l(i) \cdot card(O) + 1$$

4 Conclusion

In this paper we present a general approach for P systems with dissolution, based on the use of special symbols as promoters and inhibitors. The main result is Theorem 1, where we prove that the evolution of each transition P system Π with multiple membranes is simulated by the evolution of its "flat" counterpart Π^f. This result is a generalization of the existing construction for P systems without dissolution, as can be seen in Corollary 1.

The results presented here may be used to simplify proofs of statements involving general transition P systems by using only P systems with one membrane. However, a caveat applies: the evolution in Π^f is staggered with respect to the evolution in Π since two steps will take place in Π^f for one involving dissolution on Π. Other concerns may appear regarding the increasing number of objects and rules in the P system Π^f, according to Remark 1.

The idea of using a single membrane system to simulate P systems with multiple membranes has previously appeared in several papers. A formal presentation for (tissue) P systems without dissolution can be found in [4].

An early paper dealing with dissolution is [6]. While the paper is not directly concerned with the simulation of a multiple membrane system by a one membrane system, it presents the encoding of a multiple membrane system with dissolution into a particular kind of Petri net. The resulting Petri net has transitions which simulate rule applications and special transitions which simulate objects passing from dissolved membranes to their parents. In terms of Example 1, these special transitions simulate rules $(x, 3) \to (x, 2)$ and $(x, 2) \to (x, 1)$ for $x \in \{a, b\}$. The authors do not explain in sufficient detail the semantics of their version of Petri nets, and do not treat the case of simultaneous dissolutions. More precisely, the Petri net simulating Example 1 should have three phases in the "macro-step" in order to properly simulate the evolution of the system: one for simulating maximally parallel rule application, one for moving objects from the dissolved membrane 3 to membrane 2 and one for moving objects from the dissolved membrane 2 to membrane 1.

A recent paper presenting a flat form for P systems is [2]. The construction of this paper depends on the use of a special syntax and semantics for P systems, named *P algebra*. This semantics, while complicated, is useful in establishing various behavioural equivalences.

Acknowledgements. This research work was partially supported by the CNCSIS Grant 402/2007. We thank the reviewers for their helpful comments, as well as Artiom Alhazov, Rudi Freund and George Păun for their insightful suggestions during the CMC presentation.

References

1. Agrigoroaiei, O., Ciobanu, G.: Rewriting Logic Specification of Membrane Systems with Promoters and Inhibitors. Electronic Notes in TCS 238, 5–22 (2009)
2. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: A P Systems Flat Form Preserving Step-by-step Behaviour. Fundamenta Informaticae 87, 1–34 (2008)
3. Bottoni, P., Martín-Vide, C., Paun, G., Rozenberg, G.: Membrane Systems with Promoters/Inhibitors. Acta Informatica 38, 695–720 (2002)
4. Freund, R., Verlan, S.: A Formal Framework for Static (Tissue) P Systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 271–284. Springer, Heidelberg (2007)
5. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
6. Qi, Z., You, J., Mao, H.: P Systems and Petri Nets. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 286–303. Springer, Heidelberg (2004)

The Family of Languages Generated by Non-cooperative Membrane Systems

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

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

Abstract. The aim of this paper is to study the family of languages generated by the transitional membrane systems without cooperation and without additional ingredients. The fundamental nature of these basic systems makes it possible to also define the corresponding family of languages in terms of derivation trees of context-free grammars. We also compare this family to the well-known language families and discuss its properties. An example of a language is given which is considerably more "difficult" than those in the established lower bounds.

1 Introduction

Membrane computing is a theoretical framework of parallel distributed multiset processing. It has been introduced by Gheorghe Păun in 1998, and has been an active research area; see [14] for the comprehensive bibliography and [9],[11] for a systematic survey. Membrane systems are also called P systems.

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

– Represent words by string objects. Rather many papers take this approach, see Chapter 7 of [11], but only few consider parallel operations on words. Moreover, a tuple of sets or multisets of words is already a quite complicated structure. The third drawback is that it is very difficult to define an elegant way of interactions between strings. Polarizations and splicing are examples

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 65–80, 2010.

of that; however, these are difficult to use in applications. In this paper we focus on symbol objects.

- Represent a word by a single symbol object, or by a few objects of the form (letter,position) as in, e.g., [3]. This only works for words of bounded length, i.e., one can speak about at most finite languages.
- Represent positions of the letters in a word by nested membranes. The corresponding letters can be then encoded by objects in the associated regions, membrane types or membrane labels. Working with such a representation, even implementing a rule $a \to bc$ requires sophisticated types of rules, like creating a membrane around existing membrane, as defined in [4].
- Consider letters as digits and then view words as numbers, or use some other encoding of words into numbers or multisets. Clearly, the concept of words ceases to be direct with such encoding. Moreover, implementing basic word operations in this way requires a lot of number processing, not to speak of parallel word operations.
- Work in accepting mode, see, e.g., [5]. It is necessary to remark that in this article we are mainly speaking of non-cooperative P systems, and working in accepting mode without cooperation yields rather trivial subregular results. More exactly, such systems would accept either the empty language or only the empty word, or all the words over some subalphabet.
- Consider traces of objects across membranes. This is an unusual approach in the sense that the result is not obtained from the final configuration, but rather from the behavior of a specific object during the computation. This makes it necessary to introduce an observer, complicating the model.
- Do all the processing by multisets, and regard the order of sending the objects in the environment as their order in the output word. In case of ejecting multiple symbols in the same step, the output word is formed from any of their permutations. One can say that this approach also needs an implicit observer, but at least this observer only inspects the environment and it is, in some sense, the simplest possible one. This paper is devoted to this concept.

Informally, the family of languages we are interested in is the family generated by systems with parallel applications of non-cooperative rules that rewrite symbol objects and/or send them between the regions. This model has been introduced already in 2000, [10]. Surprisingly, this language family did not yet receive enough attention of researchers. Almost all known characterizations and even bounds for generative power of different variants of membrane systems with various ingredients and different descriptional complexity bounds are expressed in terms of REG, MAT, $ET0L$ and RE, their length sets and Parikh sets (and much less often in terms of FIN or other subregular families, or CF or CS, or those accepted by log-tape bounded Turing machines, [6], [8]). The membrane systems language family presents interest since we show it lies between regular and context-sensitive families, being incomparable with well-studied intermediate ones. As we show in the paper, the nature of $LOP_*(ncoo, tar)$ is quite fundamental, and in the same time it is not characterized in terms of well-studied families. This is why we refer to it here as *the membrane systems language family*.

This paper is based on the ideas and initial work described in [2]. We also give an example of a language which is considerably more "difficult" than the currently established lower bounds. The word "difficult" informally refers to two kinds of non-context-freeness needed to describe the language. The internal one can be captured by permutations, while the external one can be captured by an intersection of linear languages.

2 Definitions

2.1 Formal Language Preliminaries

Consider a finite set V. The set of all words over V is denoted by V^*, the concatenation operation is denoted by $\bullet$ (which is written only when necessary) and the empty word is denoted by λ. Any set $L \subseteq V^*$ is called a language. For a word $w \in V^*$ and a symbol $a \in V$, the number of occurrences of a in w is written as $|w|_a$. The permutations of a word $w \in V^*$ are $\texttt{Perm}(w) = \{x \in V^* \mid |x|_a = |w|_a \forall a \in V\}$. We denote the set of all permutations of the words in L by $\texttt{Perm}(L)$, and we extend this notation to families of languages. We use FIN, REG, LIN, CF, MAT, CS, RE to denote finite, regular, linear, context-free, matrix without appearance checking and with erasing rules, context-sensitive and recursively enumerable families of languages, respectively. The family of languages generated by extended (tabled) interactionless L systems is denoted by $E(T)0L$. For more formal language preliminaries, we refer the reader to [12].

Throughout this paper we use string notation to denote the multisets. When speaking about membrane systems, keep in mind that the order in which symbols are written is irrelevant, unless we speak about the symbols sent to the environment. In particular, speaking about the contents of some membrane, when we write $a_1^{n_1} \cdots a_m^{n_m}$ (or any permutation of it), we mean a multiset consisting of n_i instances of symbol a_i, $1 \leq i \leq m$.

2.2 Transitional P Systems

A membrane system is defined by a construct

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

O is a finite set of objects,

μ is a hierarchical structure of membranes, bijectively labeled by $1, \cdots, m$,
the interior of each membrane defines a region;
the environment is referred to as region 0,

w_i is the initial multiset in region $i, 1 \leq i \leq m$,

R_i is the set of rules of region $i, 1 \leq i \leq m$,

i_0 is the output region; when languages are considered, $i_0 = 0$ is assumed.

The rules of a membrane systems have the form $u \rightarrow v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$

are written as a subscript, and target *here* is typically omitted. In case of non-cooperative rules, $u \in O$.

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

A configuration of a P system is a construct which contains the information about the hierarchical structure of membranes as well as the contents of every membrane at a definite moment of time. The process of applying all rules which are applicable in the current configuration and thus obtaining a new configuration is called a transition. A sequence of transitions is called a computation. The computation halts when such a configuration is reached that no rules are applicable. The result of a (halting) computation is the *sequence* of objects sent to the environment (all the permutations of the symbols sent out in the same time are considered). The language $L(\Pi)$ generated by a P system Π is the union of the results of all computations. The family of languages generated by non-cooperative transitional P systems with at most m membranes is denoted by $LOP_m(ncoo, tar)$. If the number of membranes is not bounded, m is replaced by $*$ or omitted. If the target indications of the form in_j are not used, *tar* is replaced by *out*.

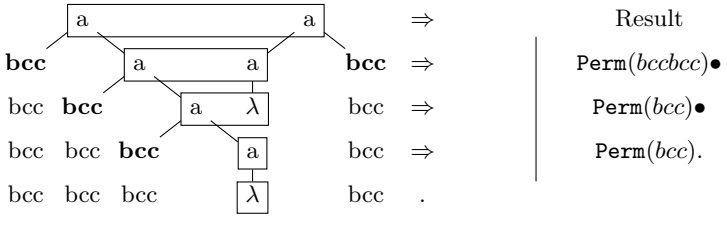


Fig. 1. An example of a computation of a P system from Example 1. The lines are only used to hint how the rules are applied.

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

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

Each of the two symbols a has a non-deterministic choice whether to be erased or to reproduce itself while sending a copy of b and two copies of c into the environment. Therefore, the contents of region 1 can remain a^2 for an arbitrary number $m \geq 0$ of steps, and after that at least one copy of a is erased. The other copy of a can reproduce itself for another $n \geq 0$ steps before being erased.

Each of the first m steps, two copies of b and four copies of c are sent out, while in each of the next n steps, only one copy of b and two copies of c are ejected. Therefore, $L(\Pi) = (\mathtt{Perm}(bccbcc))^*(\mathtt{Perm}(bcc))^*$.

3 Context-Free Grammars and Time-Yield

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

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

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

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

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

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

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

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

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

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

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

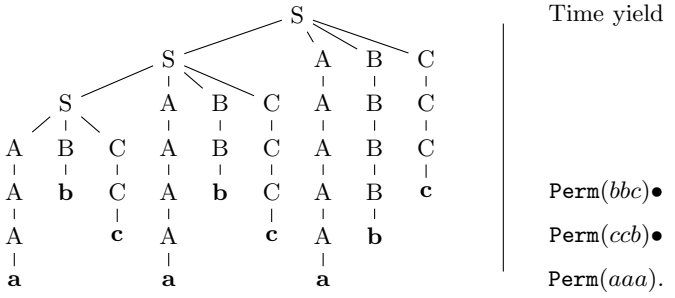


Fig. 2. An example of a derivation of the grammar from Example 2

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

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

4 The Membrane Family via the Derivation Trees of Context-Free Grammars

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

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

Proof. Consider an arbitrary transitional membrane system Π (without cooperation and without additional ingredients). The known technique of flattening the structure (this is "folklore" in membrane computing; see, e.g., [13], [7]) consists of transforming Π in the following way. Object a in region associated to membrane i is transformed into object (a, i) in the region associated to the single membrane. The alphabet, initial configuration and rules are transformed accordingly. Clearly, the configurations of the original system and the new system are bisimilar, and the output in the environment is the same. □

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

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

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

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

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

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

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

We now present a few normal forms for the context-free grammars in the context of the time yield. Note that these normal forms incorporate a number of similarities with both L systems and standard CF grammars, because level-by-level derivation in context-free grammars corresponds to the evolution in L systems. However, it is not possible to simply take existing normal forms, because the result must be preserved, and the result is defined in a different way.

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

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

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

$$(A \to \lambda) \longrightarrow (A \in E),$$
$$(A \to A_1 \cdots A_k),\ (A_1, \cdots, A_k \in E) \longrightarrow (A \in E).$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remove all non-terminals that are not productive from N, and all productions containing them. If the axiom was also removed, then $L_t(G) = \emptyset$, hence we can

take $G' = (\{S\}, T, S, \{S \to S\})$. Otherwise, compute the set $R \subseteq N$ of reachable non-terminals as closure of

$$(S \in R),$$
$$(A \in R),\ (A \to A_1 \cdots A_k) \longrightarrow (A_1, \cdots, A_k \in R).$$

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

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

5 Comparison with Known Families

Theorem 2. *[10] $LOP(ncoo, tar) \supseteq REG$.*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Proof. All inclusions and incomparabilities have been shown in or directly follow from Theorem 2, Corollary 3, Theorem 3, Corollary 2 and Corollary 1 with

Theorem 1. The strictness of the first inclusions follows from the fact that REG and $\texttt{Perm}(REG)$ are incomparable, while the strictness of the latter inclusion holds since $LOP(ncoo, tar)$ only contains semilinear languages. □

The lower bound can be strengthened as follows:

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

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

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

6 Closure Properties

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

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

Proof. Without restricting generality, we assume that the domain and range of a morphism h are disjoint (or rename the corresponding non-terminals). For a given grammar $G = (N, T, S, P)$, consider a transformation where every terminal symbol a becomes a non-terminal a' and the rules $a' \to h(a)$, $a \in T$ are added to P. It is easy to see that the new grammar generates exactly $h(L_t(G))$. □

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

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

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

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

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

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

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

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

Clearly, languages generated by any non-terminal from N must be scattered subwords of words from L, otherwise G would generate some language not in L. Thus, $L_t(G_B), L_t(G_C) \subseteq \{a, b\}^* c^*$. We now consider two cases, depending on whether $L_t(G_B) \subseteq \{a, b\}^*$.

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

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

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

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

7 A Difficult Language

In this section we give an example of a language in $LOP(ncoo, tar)$ which is considerably more "difficult" than languages in $REG \bullet \texttt{Perm}(REG)$, in the sense

informally explained below. Besides permutations of symbols sent out in the same time, it exhibits another kind of non-context-freeness. This second source of "difficulty" alone can, however, be captured as an intersection of two linear languages.

$$\begin{aligned} \Pi_D &= (\{D, D', a, b, c, a', b', c'\}, [_1 \ ]_1, D^2, R), \\ R &= \{D \to (abc)_{out} D'D', D \to (abc)_{out}, \\ &\quad D' \to (a'b'c')_{out} DD, D' \to (a'b'c')_{out}\}. \end{aligned}$$

The contents of region 1 is a population of objects D, initially 2, which are primed if the step is odd. Assume that there are k objects inside the system. At every step, every symbol D is either erased or doubled (and primed or deprimed), so the next step the number of objects inside the system will be any even number between 0 and $2k$. In addition to that, the output during that step is $\mathtt{Perm}((abc)^k)$, primed if the step is odd. Hence, the generated language can be described as

$$\begin{aligned} L(\Pi_D) = \{ \ & \mathtt{Perm}((abc)^{2k_0})\mathtt{Perm}((a'b'c')^{2k_1}) \cdots \\ & \mathtt{Perm}((abc)^{2k_{2t}})\mathtt{Perm}((a'b'c')^{2k_{2t+1}}) \\ & \mid k_0 = 1, \ 0 \le k_i \le 2k_{i-1}, \ 1 \le i \le 2t+1, \ t \ge 0\}. \end{aligned}$$

For an idea of how complex a language generated by some non-cooperative membrane system be, imagine that the skin may contain populations of multiple symbols that can (like D in the example above) be erased or multiplied (with different periods), and also rewritten into each other. The same, of course, happens in usual context-free grammars, but since the terminal symbols are collected from the derivation tree level by level instead of left to right, the effect is quite different.

Another upper bound has been recently proved.

Theorem 7. *[1] $LOP(ncoo, tar) \subseteq \mathbf{P}$.*

This result means that membrane systems languages can be parsed in polynomial time. However, the degree of such polynomials in the algorithm deciding membership problem presented in [1] depends on the number of rules and the size of the alphabet in the underlying P system.

8 Conclusions

In this paper we have reconsidered the family of languages generated by transitional P systems without cooperation and without additional control. It was shown that one membrane is enough, and a characterization of this family was given via derivation trees of context-free grammars. Next, three normal forms were given for the corresponding grammars. It was than shown that the membrane systems language family lies between regular and context-sensitive families of languages, and it is incomparable with linear and with context-free languages.

Then, the lower bound was strengthened to $REG \bullet \texttt{Perm}(REG)$. An example of a considerably more "difficult" language was given than the lower bound mentioned above. We also mention another upper bound result, i.e., membrane systems languages can be parsed in polynomial time [1].

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

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

- Clearly, $LOP(ncoo, tar) \not\supseteq MAT$. What about $LOP(ncoo, tar) \subseteq MAT$?
- Is $LOP(ncoo, tar)$ closed under arbitrary morphisms? Conjecture: no. The difficulty is to handle $h(a) = bc$ if many symbols a can be produced in the same step.
- Look for sharper lower and upper bounds.

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

References

1. Alhazov, A., Ciubotaru, C., Ivanov, S., Rogozhin, Y.: Membrane Systems Languages Are Polynomial-Time Parsable. Computer Science Journal of Moldova (2010) (submitted)
2. Alhazov, A., Ciubotaru, C., Rogozhin, Y., Ivanov, S.: The Membrane Systems Language Class. In: Eighth Brainstorming Week on Membrane Computing, Sevilla, 23–35 (2010); And LA Symposium, RIMS Kôkyûroku Series 1691, Kyoto University, pp. 44–50 (2010)
3. Alhazov, A., Sburlan, D.: Static Sorting P Systems. Applications of Membrane Computing. In: Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.) Applications of Membrane Computing. Natural Computing Series, pp. 215–252. Springer, Heidelberg (2005)
4. Bernardini, F., Gheorghe, M.: Language Generating by means of P Systems with Active Membranes. Brainstorming Week on Membrane Computing, Technical Report, 26/03, Rovira i Virgili University, Tarragona, 46–60 (2003)
5. Csuhaj-Varjú, E.: P Automata. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 19–35. Springer, Heidelberg (2005)
6. Csuhaj-Varjú, E., Ibarra, O.H., Vaszil, G.: On the Computational Complexity of P Automata. Natural Computing 5(2), 109–126 (2006)
7. Freund, R., Verlan, S.: A Formal Framework for Static (Tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 271–284. Springer, Heidelberg (2007)
8. Ibarra, O.H., Păun, G.: Characterizations of Context-Sensitive Languages and Other Language Classes in Terms of Symport/Antiport P Systems. Theoretical Computer Science 358(1), 88–103 (2006)

9. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
10. Păun, G., Rozenberg, G., Salomaa, A.: Membrane Computing with an External Output. Fundamenta Informaticae 41(3), 313–340 (2000)
11. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
12. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1-3. Springer, Heidelberg (1997)
13. Qi, Z., You, J., Mao, H.: P Systems and Petri Nets. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 286–303. Springer, Heidelberg (2004)
14. P systems webpage, http://ppage.psystems.eu/

Polymorphic P Systems

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

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

Abstract. Membrane computing is a formal framework of distributed parallel computing. In this paper we introduce a variant of the multiset rewriting model where the rules of every region are defined by the contents of interior regions, rather than being explicitly specified in the description of the system. This idea is inspired by the von Neumann's concept of "program is data" and also related to the research direction proposed by Gh. Păun about the cell nucleus.

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 this paper we define yet another, relatively powerful, extension to the model, which allows the system to dynamically change the set of rules, not limited to some finite prescribed set of candidates. There are three motives for this extension. First, our experience shows that "practical" problems need "more" computing potential than just computational completeness. Second, we attempt to import a very important computational ingredients into P systems, this time from the conventional computer science. Third, this extension correlates with the biological idea that different actions are carried out by different objects, which too can be acted upon. (This last idea was also considered in, e.g., [6] and [1], but there one represented each rule by a single objects, therefore all rules were still prescribed, though not their multiplicities.) Let us first explain these motives.

Most papers of the field belong to the following categories: 1) introducing different models and variants, 2) studying the computational power of different models depending on what ingredients are allowed and on the descriptional complexity parameters, 3) studying the computational efficiency of solving intractable problems (supercomputing potential) depending on the ingredients,

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 81–94, 2010.

4) using membrane computing to represent and model various processes and phenomena, including but not limited to biology, 5) other applications.

There is a surprisingly big gap between the sets of ingredients needed to fulfill requirements in directions 2, 3, and the sets of ingredients demanded by other applications. For instance, very weak forms of cooperation between objects are often enough for the computational completeness, but many "practical" problems cannot be solved in a satisfactory way under the same limitations. This leads to the following question.

1.1 What Is Implicitly Required in Most "Practical" Problems?

We will mention just a few of these requirements below.

- **Determinism** or at least confluence. Clearly, the end user wants to obtain the answer to the specified problem in a single run of a system instead of examining infinitely many computations. This is a strong constraint, e.g., catalytic P systems and P systems with minimal symport/antiport are universal, while in the deterministic case non-universality is published for the first ones and claimed for the latter ones. Informally speaking, less computational power is needed to just compute the result than it is to also enforce choice-free behavior of the system.
- **Input/output**. Most of the universality results are formulated as generating languages or accepting sets of vectors, or in an even more restricted setup. There is no need to deal with input in the first case, and in the latter case the final configuration itself is irrelevant (except yes or no in case of the efficiency research). On the other side, both input and output are critical for most applications.
- **Representation**. Clearly, any kind of discrete information can be encoded in a single integer in some consistent way. However, a much more transparent data representation is typically required; even the intermediate configurations in a computation are expected to reflect a state of the object in the problem area.
- **Efficiency**. Suppose numbers are represented by multiplicities of certain objects. The number of steps needed to multiply two numbers by plain (cooperative) multiset processing is proportional to the result. If the multiset processing can be controlled by promoters/inhibitors/priorities, then the number of steps needed for multiplication is proportional to one of the arguments. However, many applications would ask for a multiplication to be performed in a constant number of steps. Similar problems appear for string processing.
- **Data structures**. Membrane computing deals with multisets distributed over a graph, while conventional computers provide random memory access and pointer operations, allowing much more complex structures to be built.

Some of these implicit requirements originate because the user wants a solution which is at least as good as the one that can be provided by conventional computers. We hope that the explanations of the above list have convinced the reader that this is often a challenge.

1.2 Program is Data. Cell Nucleus

In this paper we try to introduce another feature into the membrane computing. This time the inspiration is not biological, but rather is from the area of conventional computing. Suppose we want to be able to manipulate the rules of the system during its computation. A number of papers has been written in this direction (see, e.g., GP systems [6], rule creation [3], activators [1], inhibiting/deinhibiting rules [4] and symport/antiport of rules [5]), but in most of them the rules are predefined in the description of the system.

The most natural way to manipulate the rules is to represent them as data, treat this data as rules, and manipulate it as usual in P systems, in the spirit of von Neumann's approach. In membrane systems, the data consists of multisets, so objects should be treated as description of the rules. Informally, a rule j in a region i can be represented by the contents of membranes jL and jR inside i.

For instance, $\boxed{\begin{array}{c}1 : ab \to ac \\ 2 : a \to d \\ abbb\end{array}}_s$ becomes $\boxed{\boxed{ab}_{1L}\ \boxed{ac}_{1R}\ \boxed{a}_{2L}\ \boxed{d}_{2R}\ abbb}_s$.

Changing the contents of regions jL and jR results in the corresponding change of the rule j. The next section illustrates this effect in Figure 1 and gives the formal definitions. We call such P systems polymorphic, by analogy with polymorphic, or self-modifying computer programs.

At the same time, if a membrane system is an abstraction inspired by the biological cell, one can view inner regions as an abstraction inspired by the cell nucleus; their contents correspond to the genes encoding the enzymes performing the reactions of the system. The simplicity of the proposed model is that we consider the natural encoding, i.e., no encoding at all: the multisets describing the rules are represented by exactly themselves. Therefore, we are addressing a problem informally stated by Gh. Păun in Section "Where Is the Nucleus?" of [7] by proposing a computational variant based on one simple difference: the rules are taken from the current configuration rather than from the description of the P system itself.

The idea of a nucleus was also considered in [9], but such a presentation had the following drawbacks. First, one described the dynamics of the rules in a high-level programming language (good for simulators, but otherwise too powerful extension having the power of conventional computers directly built into the definition). Second, this dynamics of the rules did not depend on the actual configuration of the membrane system (no direct feedback from objects to rules). In the model presented in this paper, the dynamics of rules is defined by exactly the same mechanism as the standard dynamics of objects.

2 Definitions

We refer the reader to [8] for the standard preliminaries of membrane computing. We denote the family of recursively enumerable sets of non-negative integers by NRE.

We define a polymorphic P system as a tuple

$$\Pi = (O, T, \mu, w_s, w_{1L}, w_{1R}, \cdots, w_{mL}, w_{mR}, \varphi, i_{out}),$$

where O is a finite alphabet, μ is a tree structure consisting of $2m+1$ membranes, bijectively labeled by elements of $H = \{s\} \cup \{iL, iR \mid 1 \leq i \leq m\}$ (the skin membrane is labeled by s; we also require for $1 \leq i \leq m$ that the parent membrane of iL is the same as the parent membrane of iR), w_i is a string describing the contents of region i, $1 \leq i \leq m$, and φ is a mapping from $\{1, \cdots, m\}$ to the features of the rules described below. The set $T \subseteq O$ describes the output objects, while $i_{out} \in H \cup \{0\}$ is the output region (0 corresponds to the environment).

Notice that the rules of a P system are not explicitly given in its description. Essentially, such a system has m rules, and these rules change as the contents of regions other than skin changes. Initially, for $1 \leq i \leq m$ rule $i : w_{iL} \to (w_{iR}, \varphi(i))$ 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 initial ones.

In what follows we mainly consider a single feature, i.e., target indications. In this case, the range of φ is $Tar = \{in_i \mid i \in H\} \cup \{here, out\}$. We denote the class of all polymorphic P systems with cooperative rules and target indications and at most k membranes by

$$OP_k(polym_{+d}(coo), tar).$$

In the notation above, the number k is replaced by $*$ or omitted if no bound is specified. The subscript $+d$ means that the rules can be disabled; we write $-d$ instead, if w_{iL} is never empty for $1 \leq i \leq m$ during any computation. We prefix this notation with D if we restrict the class to the deterministic systems (for every input if it is specified, see below).

A computation is a sequence of configurations starting in the initial configuration, corresponding to the transitions induced by non-deterministic maximally parallel application of rules; it is called halting if no rules are applicable to the last configuration. In the latter case the multiset of objects from T in region i_{out} is called the result.

If we want to compute instead of generating, we extend the tuple Π by the description of the input as follows. In the definition of the P system, we insert the input alphabet $\Sigma \subset O$ after O and we insert the input region i_{in} after φ. In this case, the input multiset over Σ is added to $w_{i_{in}}$ before the computation starts. If we want to accept instead of computing, we remove T and i_{out} from the description of the P system; the input is considered accepted if and only if the system may halt. If we want to decide instead of computing, we construct a system that always halts with either `yes` or `no` in the output region, such that this answer uniquely depends on the input; the input is accepted if and only if the answer is `yes`. Speaking about the time complexity is more appropriate for deciding than for accepting.

The set of numbers or vectors generated by a P system Π is denoted by $N(\Pi)$ or $Ps(\Pi)$, respectively. In the accepting case, we write $N_a(\Pi)$ or $Ps_a(\Pi)$. In the deciding case, we write $N_d(\Pi)$ or $Ps_d(\Pi)$. If the computation of Π is deterministic for every input, then the partial function computed by Π is denoted by $f(\Pi)$. In this way, the entire class of polymorphic P systems with cooperative rules and target indications, allowing disabled rules, with at most k membranes, defines a family of sets of numbers, of sets of vectors or of functions, respectively denoted by

$$NOP_k(polym_{+d}(coo), tar),\ \ PsOP_k(polym_{+d}(coo), tar),$$
$$fDOP_k(polym_{+d}(coo), tar).$$

In a similar way it is possible to replace cooperative rules with a more restricted set, remove target indications or add more features to the polymorphic P systems, modifying the notation accordingly. It is even possible to consider completely different rules instead of rewriting, e.g., symport/antiport rules, but we do not address such a topic here.

We illustrate the definitions by the following example.

Example 1. A P system with a superexponential growth.

$$\Pi_1 = (\{a\}, \{a\}, \mu, a, a, a, a, a, a, aa, \varphi, 1), \text{ where}$$
$$\mu = [\ [\ ]_{1L}[\ [\ ]_{2L}[\ [\ ]_{3L}[\ ]_{3R}\ ]_{2R}\ ]_{1R}\ ]_s,$$
$$\varphi(i) = here,\ 1 \le i \le 3.$$

Naturally, contents of membranes $1L$, $2L$, $3L$ is never changed because they are elementary and no rules have the corresponding target indications, and their initial contents is a, so the system is non-cooperative, and the rules are never disabled. Since only one rule acts in each of the regions s, $1R$, $2R$, the system is deterministic. From all above we conclude that $\Pi_1 \in DOP_7(polym_{-d}(ncoo))$, a quite restricted class.

This system never halts. Its interesting aspect, however, is the growth of the number of objects in the skin. We claim that at step n the skin contains $2^{n(n-1)(n-2)/6}$ objects, so the growth function is an exponential of a polynomial. Indeed, this is not difficult to see by starting from the elementary membranes and going outside.

The contents of $3R$ is aa and it never changes. Region $2R$ initially contains a and undergoes rule $a \to aa$ every step, so its contents at step n is a^{2^n}. Region $1R$ initially contains a and undergoes rule $a \to a^{2^n}$ at step n, so its contents at step n is $a^{2^{n(n-1)/2}}$. The skin originally contains a and at step n rule $a \to a^{2^{n(n-1)/2}}$ is applied, so its contents at step n is $a^{2^{n(n-1)(n-2)/6}}$, see Figure 1 for the actual illustration of the computation and for the proof of the result.

This growth is faster than that of any non-polymorphic P systems, which is bounded by the exponential Ic^n, where I is the initial number of objects in the system and c is the maximum ratio for all rules of the right side size and its left side size. It is not difficult to see that the growth function of a polymorphic

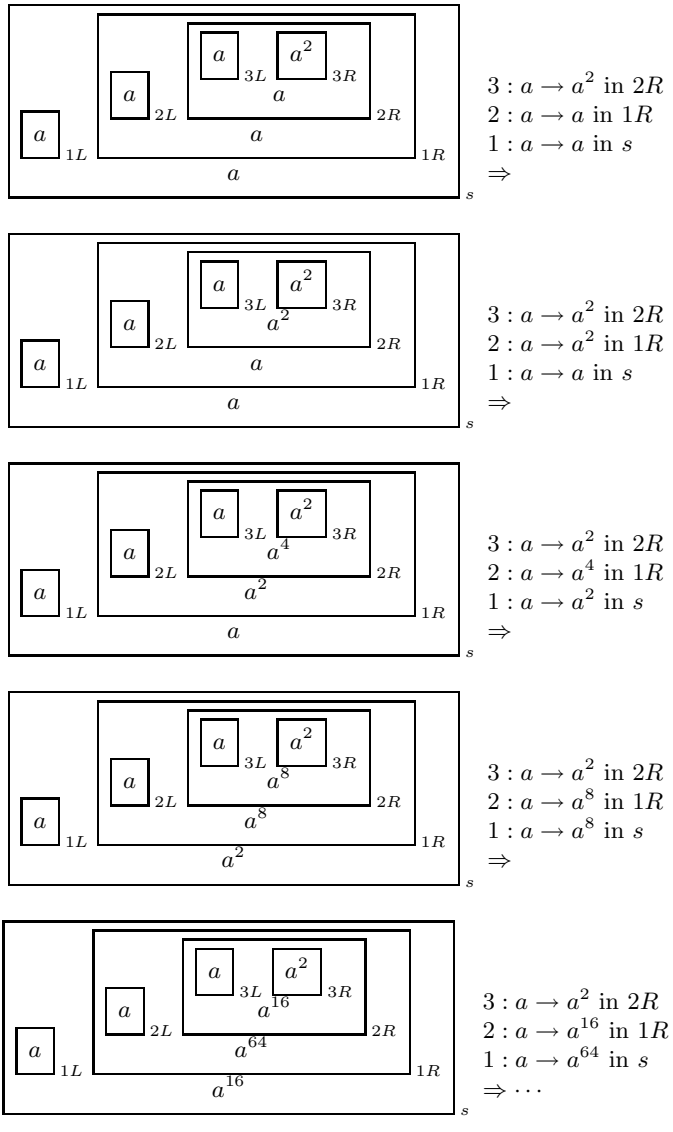


Fig. 1. The computation of Π_1 from Example 1. If the number of objects a in regions $3R$, $2R$ $1R$, s at step n is (x_n, y_n, z_n, t_n), respectively, then $(x_0, y_0, z_0, t_0) = (2, 1, 1, 1)$ and $(x_{n+1}, y_{n+1}, z_{n+1}, t_{n+1}) = (x_n, y_n x_n, z_n y_n, t_n z_n)$.

Following just this quadruple, the computation can be represented as $(2, 1, 1, 1) \Rightarrow (2, 2, 1, 1) \Rightarrow (2, 4, 2, 1) \Rightarrow (2, 8, 8, 2) \Rightarrow (2, 16, 64, 16) \Rightarrow (2, 32, 1024, 1024) \Rightarrow (2, 64, 32768, 1048576) \Rightarrow \cdots$.

The exponents of the closed form formula $(2, 2^n, 2^{n(n-1)/2}, 2^{n(n-1)(n-2)/6})$ can be verified as follows. $n + 1 = n + 1$, $(n + 1)n/2 = n(n - 1)/2 + n$, $(n + 1)n(n - 1)/6 = n(n - 1)(n - 2)/6 + n(n - 1)/2$.

P system without target indications is bounded by $Ic^p(n)$, where I and c are defined as above and p is a polynomial whose degree equals the depth of the membrane structure minus one.

3 Results

As long as full cooperation is allowed, the universality of polymorphic P system is not difficult to obtain, even without the actual polymorphism (i.e. without ever modifying rules) and without the use of target indications. The upper bound on the number of membranes needed is one plus twice the number of rules, because in the polymorphic P systems the rules can only be represented by pairs of membranes. We recall that in [2] one presents a strongly universal P system with 23 rules. Hence, the following theorem holds.

Theorem 1. $NOP_{47}(polym_{-d}(coo)) = NRE.$

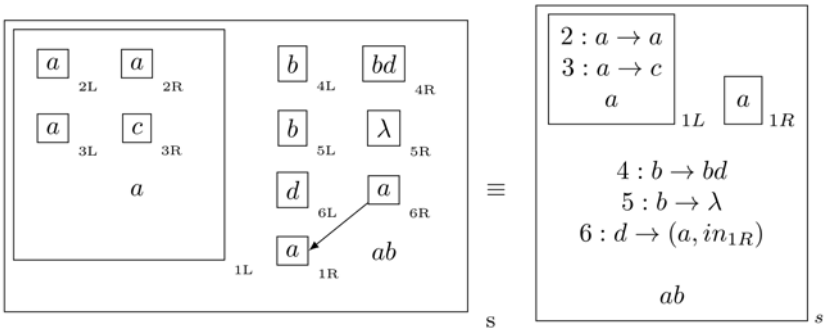
Proof. The claim is fulfilled by taking the one-membrane construction from the main result in [2] and replacing each of the 23 rules by two membranes containing the left-hand side and the right-hand side of that rule.

In the rest of the paper we focus on the efficiency of computations performed by polymorphic P systems, using the time complexity terms. We devote special attention to fast generating and deciding factorials, because they best illustrate constant-time multiplication where the factors are not known in advance and are even changing during the computation. First, we present a non-cooperative system generating "slightly" more than factorials, using target indications. It is a bit more complicated than Π_1 because, firstly, we need to multiply by numbers that grow linearly, and secondly, we want the system to halt.

Example 2. A polymorphic P system from $OP_{13}(polym_{-d}(ncoo), tar)$ which generates $\{n! \cdot n^k \mid n \geq 1, k \geq 0\}$.

$$\Pi_2 = (\{a, b, c, d\}, \{a\}, \mu, ab, a, a, a, a, a, c, b, bd, b, \lambda, d, a, \varphi, 1), \text{ where}$$
$$\mu = [\,[\,[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}\,]_{1L}[\,]_{1R}[\,]_{4L}[\,]_{4R}[\,]_{5L}[\,]_{5R}[\,]_{6L}[\,]_{6R}\,]_s,$$
$$\varphi(i) = here,\ 1 \leq i \leq 5,\ \varphi(6) = in_{1R}.$$

The initial configuration can be graphically represented as shown below. In fact, such a graphical representation gives a complete description of Π_2 except the output alphabet and the output region. The target indication of a rule (here rule 6 in $1R$) may be indicated by an arrow, in this case from $6R$ to $1R$ (keeping in mind that the reactants of the rule are taken from the parent region of the membranes describing the rule, in this case, from region 1). At the right we give a simplified representation of the same system by replacing pairs of membranes with constant contents by the rules written explicitly (this is just a different representation, so-called "syntactic sugar", and we still count such rules as pairs of membranes). Rule 1 is not written with the rule syntax because the contents of both $1L$ and $1R$ will change.



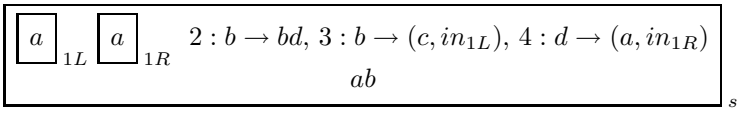
The essence of the functioning of Π_2 is the following. Rules 4 and 6 lead to incrementation of the number of copies of a in $1R$ (the number of copies of a in the skin does not change during the first two steps). The system will apply rule 4 for $n-1 \geq 0$ times and then rule 5 (applying rule 5 is necessary for the system to halt). Suppose that all this time rule 2 has been applied in region $1L$. Then, the number of objects in region $1R$ will grow linearly, and subsequent applications of a dynamic rule $1 : a \to a^i$, $1 \leq i \leq n$ will produce $a^{n!}$ in the skin. After that, the number of objects a in the skin will be multiplied by n until rule 3 is applied, because $1 : c \to a^n$ will be no longer applicable, halting with the skin only containing objects a their number being an arbitrary number of the form $n! \cdot n^k$. Now assume that rule 3 has been applied earlier, effectively stopping the multiplication of the number of objects a in the skin before the incrementation of objects a in $1R$ is finished. In that case the multiplicity of objects a in the skin will be just a factorial of a smaller number, and the system will evolve by application of rules 4, 6 until rule 5 is applied, without affecting the result. Notice that the time complexity (understood as the shortest computation producing the corresponding result) of generating $n! \cdot n^k$ is only $n + k + 1$.

To generate exactly $\{n! \mid n \geq 1\}$ we need to stop the multiplication when we stop the increment. This seems impossible without cooperative rules.

Example 3. A P system from $OP_9(polym_{-d}(coo), tar)$ generating $\{n! \mid n \geq 1\}$.

$$\Pi_3 = (\{a, b, c, d\}, \{a\}, \mu, ab, a, a, b, bd, b, c, d, a, \varphi, 1), \text{ where}$$
$$\mu = [\,[\,[\,]_{1L}[\,]_{1R}[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}[\,]_{4L}[\,]_{4R}\,]_s,$$
$$\varphi(i) = here,\ 1 \leq i \leq 2,\ \varphi(3) = in_{1L},\ \varphi(4) = in_{1R}.$$

This system is very similar to Π_2. There are only the following differences. First, rules $a \to a$ and $a \to c$ are removed from region $1L$. Second, instead of erasing b in the skin, the corresponding rule sends object c to region $1L$, which stops both increment (b is erased) and multiplication ($1 : ac \to a^n$ is not applicable in the skin). Ironically, this system never applies any non-cooperative rule, but the non-cooperative feature seems unavoidable in order to stop the computation in the synchronized way. A compact graphical representation of Π_3 is given below.



Now we proceed to describing a P system generating $\{2^{2^n} \mid n \geq 0\}$ in $O(n)$ steps. Since the growth of polymorphic P systems without target indications is bounded by exponential of polynomials, the system below grows faster than any of them. Moreover, it produces the above mentioned result by halting.

It is also worth noting that even polymorphic P systems cannot grow faster than exponential of exponential in linear time, because if a system has $n+n+1 > 3$ objects at some step, then it cannot have more than n^2+n+1 objects in the next step. Indeed, consider that some rule r is applied for n times; let its left side contain x objects and let its right side contain y objects. Then, $x+y$ objects are needed to describe the rule and they transform nx other objects into ny objects. It is not difficult to see that the growth is maximal if $x = 1$ and $y = n$. Since n^2+n+1 is less than the square of $n+n+1$, and iterated squaring yields the growth which is exponential of exponential, it is not possible to grow faster. The system below grows three times slower than this bound.

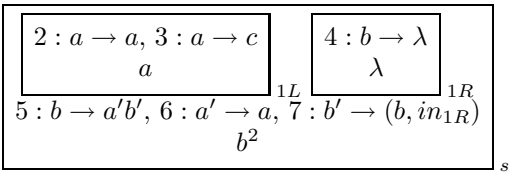
Example 4. A P system from $OP_{15}(polym_{-d}(ncoo), tar)$ generating numbers from $\{2^{2^n} \mid n \geq 0\}$ in $3n+2$ steps.

$$\Pi_4 = (\{a, b, a', b', c\}, \{a\}, \mu, b^2, a, \lambda, a, a, a, c, b, \lambda, b, a'b', a', a, b', b, \varphi, 1), \text{ where}$$

$$\mu = [\,[\,[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}\,]_{1L}[\,[\,]_{4L}[\,]_{4R}\,]_{1R} \prod_{i=5}^{7} ([\,]_{iL}[\,]_{iR})\,]_s,$$

$$\varphi(i) = here,\ 1 \leq i \leq 6,\ \varphi(7) = in_{1R}.$$

The desired effect is obtained by iterated squaring. By rules 5, 6, 7, in two steps each copy of b in the skin changes into a and also sends a copy of b in region $1R$. In the next step, if region $1L$ still contains an a, each copy of a in the skin is replaced by the contents of region $1R$, and the process continues. Therefore, if we had b^k in the skin at some step, then in two steps we will have a^k in the skin and rule 1 will be of the form $a \to b^k$, yielding b^{k^2} in the third step. The iteration continues while rule 2 is being applied in region $1L$. When rule 3 is applied, the cycle stops because rule $1 : c \to b^k$ will not be applicable, and the result is given as the multiplicity of objects a in the skin. Clearly, $2 = 2^{2^0}$ and $2^{2^{n+1}} = (2^{2^n})^2$, so the systems generates 2^nth powers of 2. We underline that no cooperation was used in this case. A compact graphical representation of this system is shown below.



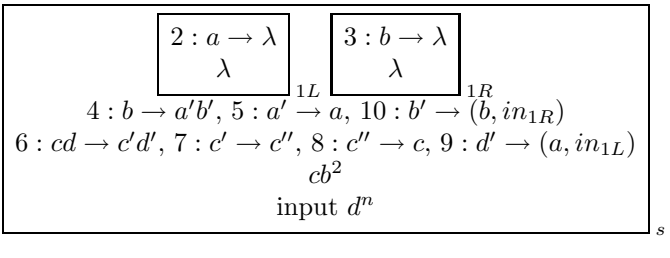
We remind the reader that the picture above represents a system with 15 membranes because the rules notation is simply a compact way to represent pairs of membranes. Note that one rule could have been saved if the right side of the rule were allowed to have objects with different target indications, but this issue does not affect the computational power, only the number of rules, whereas the definitions are much simpler. Another rule could be saved at the price of using a cooperative rule to stop the computation instead of rules 2 and 3, like in the previous example.

We now proceed to tasks which are more difficult than generating, namely, deciding a set of numbers or computing a function in a deterministic way. We illustrate the first case by modifying the previous example. We use an additional ingredient compared to the previous systems: we rely on disabling a rule by emptying the region describing its left side. Although we expect that this ingredient does not change the computational power of the systems, we use it in order to have smaller constructions.

Example 5. A deterministic P system from $OP_{15}(polym_{+d}(coo), tar)$ computing the function $n \longrightarrow 2^{2^n}$ in $3n+2$ steps.

$$\begin{aligned}\Pi_5 = (&\{a, b, a', b', c, d, d'\}, \{d\}, \{a\}, \mu, cb^2, \lambda, \lambda, a, \lambda, b, \lambda,\\ &b, a'b', a', a, cd, c'd', c', c'', c'', c, d', a, b', b, \varphi, 1, 1), \text{ where}\\ \mu = [\,[\,[\,[\,]_{2L}[\,]_{2R}\,]_{1L}[\,[\,]_{3L}[\,]_{3R}\,]_{1R} &\prod_{i=4}^{10}([\,]_{iL}[\,]_{iR})\,]_s,\\ \varphi(i) = here,\ 1 \le i \le 8,\ &\varphi(9) = in_{1L},\ \varphi(10) = in_{1R}.\end{aligned}$$

This system works like Π_4 from the previous example. We only focus on the differences. The previous system used non-deterministic choice between rules 2 and 3 to continue the computation or to stop it. In this case, squaring stops by itself due to the rule $2 : a \to \lambda$, so producing object a in region $1L$ activates one squaring. The most important difference is that the number n is given as input into the skin, by the multiplicity of objects d. Moreover, besides two copies of b the skin initially contains an object c, responsible for counting until n by consuming objects d and activating the squaring routine the corresponding number of times. The cycle takes 3 steps, see rules 6, 7, 8, 9. When object c has no more copies of d to consume, the result is obtained as the multiplicity of objects a in the skin. We show a compact graphical representation of Π_5 below.



Note that this system uses cooperation for counting and disabling the rules for easier control. We leave it as an exercise for the reader to construct a P system Π_5' computing the same function without disabling rules. Hint: as long as objects a only appear in the skin every third step, there is no need to disable rule 1 while the computation is in progress. Object c can deterministically subtract d and perform its appearance checking. Finally, when there are no copies of d in the skin, moving c into $1L$ will make rule 1 inapplicable without the need to disable it by emptying its left side.

Now we give an example of a P system deciding a set of numbers. It works deterministically and produces an object `yes` or `no` in the skin, depending on whether the input number belongs to the specified set. We also emphasize its time complexity.

Example 6. A deterministic P system from $OP_{37}(polym_{-d}(coo), tar)$ deciding the set $\{n! \mid n \geq 1\}$. A number $k \leq n!$ is decided in at most $4n$ steps, i.e., in a sublogarithmic time with respect to k.

$$\begin{aligned}\Pi_6 = (&\{a, b, c_0, c_1, c_2, A, A', B, B', p_0, p_1, p_2, p_3, \texttt{yes}, \texttt{no}\}, \{a\}, \{\texttt{yes}, \texttt{no}\}, \mu,\\ &p_0c_0, a^2, b, b, a, c_1, c_2, c_2, \lambda, p_0, AABp_1, Aa, A'a, Bb, B'b, p_1, p_2,\\ &p_2B'AA, p_3d, p_2B'A'A, f\texttt{no}, p_2B'A'A', f\texttt{no}, p_2BAA, f\texttt{no},\\ &p_2BA'A, f\texttt{yes}, p_2BA'A', f\texttt{no}, p_3, p_0c_0, c_0, c_1, d, a, f, f, \varphi, 1, 1), \text{ where}\\ \mu = [\,[\,\,]_{1L}[\,[\,\,]_{3L}[\,\,]_{3R}[\,\,]_{4L}[\,\,]_{4R}]_{2L}[\,\,]_{2R} &\prod_{i=5}^{18}([\,\,]_{iL}[\,\,]_{iR})\,\,]_s,\\ \varphi(i) = here,\ 1 \leq i \leq 15,\ &\varphi(16) = in_{2L},\ \varphi(17) = \varphi(18) = in_{1L}.\end{aligned}$$

The work of Π_6 consists of iterated division of the input a^k. Each cycle consists of 4 steps. The role of object c_0 is to enter into $2L$ by rule 16, thus preventing rule $2 : b \to a$ to work during the second and the third step of the cycle ($bc_1 \to a$ is not applicable, changing by rule 3 to $bc_2 \to a$, which is also not applicable, and then being restored by rule 4).

Object p_0 marks the steps and produces the necessary objects for checking some numbers, and finally produces symbols to increment the divisor or to modify the dividing rule to stop the computation, and give the answer, as follows. Suppose that the input is a^k. In the first step, p_0 changes into p_1, also producing checkers AAB. In the same time, the number k will be divided by n (initially $n = 2$) by rule $1 : a^n \to b$, changing a^k into $b^x a^y$, where x is the quotient and y is the remainder.

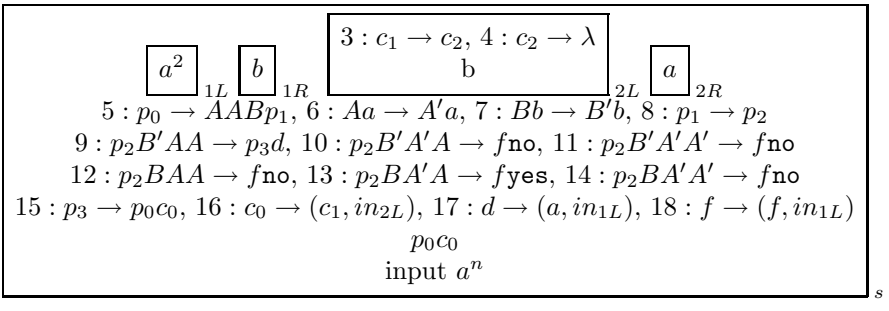
In the second step, p_1 changes into p_2, waiting for the checkers. The role of the checking rules $6 : Aa \to A'a$ and $7 : Bb \to B'b$ is to test the multiplicity of the remainder and the quotient, respectively. Hence, object B will be primed if $x > 0$. Notice that since there are two copies of A in the system, the number of symbols A that will be primed is $\min(y, 2)$. Thus, there are 6 combinations of symbols A and B, primed or not.

In the third step, we distinguish two special cases. If $x > 0$ and $y = 0$, then the input is a multiple of the currently computed factorial, and we proceed to

the next iteration by rule $9 : p_2B'AA \to p_3d$. If $x = 0$ and $y = 1$, then the input is equal to the previously computed factorial, and the system gives the positive answer by the rule $13 : p_2BA'A \to f\texttt{yes}$. Four other combinations correspond to detecting that the input is not equal to a factorial of any number (two cases correspond to non-zero quotient and non-zero remainder, the third case corresponds to the input being zero, and the last case corresponds to a multiple of some factorial which is smaller than the next factorial), so $f\texttt{no}$ is produced.

In the fourth step, rule $2 : b \to a$ is used, so the quotient is ready to be divided again. Object f is used to stop the computation by rule 18, since rule $1 : a^n f \to b$ is not applicable. In case we proceed to the next iteration, the role of object d is to increment the multiplicity n of objects a in region $1L$, and object p_3 changes back to p_0 and produces a new copy of c_0 for the next cycle.

Below is a compact graphical representation of Π_6.



We summarize some of the results we obtained as follows.

Theorem 2. *There exist*

- *A strongly universal P system from* $OP_{47}(polym_{-d}(coo))$;
- *A P system* $\Pi_1 \in DOP_7(polym_{-d}(ncoo))$ *with a superexponential growth;*
- *A P system* $\Pi_2 \in OP_{13}(polym_{-d}(ncoo), tar)$ *such that* $N(\Pi_2) = \{n! \cdot n^k \mid n \geq 1,\ k \geq 0\}$ *and the time complexity of generating* $n! \cdot n^k$ *is* $n + k + 1$;
- *A P system* $\Pi_3 \in OP_9(polym_{-d}(coo), tar)$ *such that* $N(\Pi_3) = \{n! \mid n \geq 1\}$ *and the time complexity of generating* $n!$ *is* $n + 1$;
- *A P system* $\Pi_4 \in OP_{15}(polym_{-d}(ncoo), tar)$ *such that* $N(\Pi_4) = \{2^{2^n} \mid n \geq 0\}$ *and the time complexity of generating* 2^{2^n} *is* $3n + 2$;
- *A P system* $\Pi'_5 \in DOP_*(polym_{-d}(coo), tar)$ *such that* $f(\Pi_5) = (n \longrightarrow 2^{2^n})$ *and the time complexity of computing* $n \longrightarrow 2^{2^n}$ *is* $O(n)$;
- *A P system* $\Pi_6 \in DOP_*(polym_{-d}(coo), tar)$ *such that* $N_d(\Pi_6) = \{n! \mid n \geq 1\}$ *and the complexity of deciding any number* k, $k \leq n!$ *does not exceed* $4n$.

Moreover, polymorphic P systems can grow faster than any non-polymorphic P systems, whereas even non-cooperative polymorphic P systems with targets can grow faster than any polymorphic P systems without targets.

4 Discussion

We proposed a variant of the rewriting model of P systems where the rules are represented by objects of the system itself and thus can dynamically change. This yields a mechanism whose idea is similar to the idea of the functioning of the cell nucleus (i.e., DNA represent the proteins performing certain functions on the objects including DNA), except our formalism is more elegant mathematically because of its simplicity and because we only used a trivial encoding (which is no encoding at all, except the left and right parts of the rule are given in dedicated membranes).

This variant also has a number of connections to the conventional computing, since the "program" can be changed by manipulating data (cf. von Neumann architecture vs Harvard architecture). A number of possible extensions is suggested in the Definition section of the paper.

Polymorphic P systems are universal (with 47 membranes) because non-polymorphic P systems are universal. While the growth of non-polymorphic P systems is bounded by exponential, polymorphic P systems without target indications can grow faster, bounded by an exponential of polynomials, and polymorphic P systems with target indications can grow even faster, bounded by an exponential of exponentials.

Non-cooperative polymorphic P systems can generate non-context-free sets of numbers. Cooperative polymorphic P systems can multiply numbers in constant time and generate factorials of n or exponentials of exponentials of n in time $O(n)$, which is a very important advantage over non-polymorphic P systems.

An especially interesting case is that of deciding if the input belongs to a given set, e.g., $\{n! \mid n \geq 1\}$. While non-polymorphic P systems cannot even grow with factorial speed, not to speak about halting or verifying the input, we have shown that polymorphic P systems can decide factorials in time $O(n)$. This implies that there exist infinite sets of numbers that are accepted in a time which is sublinear with respect to the size of the input in binary representation (without cheating by only examining a part of the input to accept).

Many questions are left open, we mention three questions here. First, we find it particularly interesting what is the exact characterization of the most restricted classes we defined, like $OP_*(polym_{-d}(ncoo))$. On the other hand, it seems interesting how the (general classes of) polymorphic P systems can solve the problems of real applications which non-polymorphic P system are not suitable for. Another question is whether the polymorphic P systems can effectively use superexponential growth and dynamics of rule description to solve intractable problems in polynomial time without dividing or creating membranes. Conjecture: no, because the total number of rules (counting rules in different regions as different) cannot grow.

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

References

1. Alhazov, A.: A Note on P Systems with Activators. In: Păun, G., Riscos-Núñez, A., Romero-Jimenez, A., Sancho-Caparrini, F. (eds.): Second Brainstorming Week on Membrane Computing, RGNC Report 01/2004, University of Sevilla, pp. 16–19 (2004)
2. Alhazov, A., Verlan, S.: Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. Technical Report, 862, Turku Centre for Computer Science, Turku (2008), http://tucs.fi and arXiv:1009.2706v1 [cs.FL] (September 14, 2010), http://arxiv.org/abs/1009.2706
3. Arroyo, F., Baranda, A., Castellanos, J., Păun, G.: Membrane Computing: The Power of (Rule) Creation. Journal of Universal Computer Science 8(3), 369–381 (2002)
4. Cavaliere, M., Ionescu, M., Ishdorj, T.-O.: Inhibiting/de-inhibiting Rules in P systems. In: Preproceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, pp. 174–183 (2004)
5. Cavaliere, M., Genova, D.: P systems with Symport/Antiport of Rules. Journal of Universal Computer Science 10(5), 540–558 (2004)
6. Freund, R.: Generalized P-systems. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 281–292. Springer, Heidelberg (1999)
7. Păun, G.: Research Topics in Membrane Computing. In: Gutiérrez-Naranjo, M.A., Păun, G., Riscos-Núñez, A., Romero-Campero, F.J. (eds.) Fourth Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, vol. II, pp. 235–252 (2006)
8. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
9. Ştefănescu, G., Şerbănuţa, T., Chira, C., Roşu, G.: P Systems with Control Nuclei. In: Preproceedings of the Tenth Workshop on Membrane Computing (WMC10), Curtea de Argeş, pp. 361–365 (2009)
10. P systems webpage, http://ppage.psystems.eu/

A Small Universal Splicing P System

Artiom Alhazov[1,2], Yurii Rogozhin[1], and Sergey Verlan[3,1]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău 2028 Moldova
{artiom,rogozhin}@math.md
[2] IEC, Department of Information Engineering
Graduate School of Engineering, Hiroshima University
Higashi-Hiroshima 739-8527 Japan
[3] LACL, Département Informatique, Université Paris Est
61, av. gén de Gaulle, 94010, Créteil, France
verlan@univ-paris12.fr

Abstract. In this article we present a universal splicing P system with 6 rules. Thus we improve the previous result that used 8 rules and lower the possible value for the boundary between the universality and non-universality for such systems.

1 Introduction

Head splicing systems (H systems) were one of the first theoretical model of bio-molecular computing (DNA-computing) and they were introduced by T.Head [5]. The molecules from biology are replaced by words over a finite alphabet and the chemical reactions are replaced by a *splicing* operation. An H system specifies a set of rules used to perform a splicing and a set of initial words or axioms. The computation is done by applying iteratively the rules to the set of words until no more new words can be generated. This corresponds to a bio-chemical experiment where one has enzymes (splicing rules) and initial molecules (axioms) which are put together in a tube and one waits until the reaction stops.

H systems are not very powerful, so, a lot of other models introducing additional control elements were proposed (see [9] for an overview).

Another extension of H systems was done using the framework of P systems [8], see also [4] and [10]. In a formal way, splicing P systems can be considered like a graph, whose nodes contain sets of strings and sets of splicing rules. Every rule permits to perform a splicing and to send the result to some other node.

Since splicing P systems generate any recursively enumerable languages, it is clear that there are universal splicing P systems. Like for small universal Turing machines, we are interested by such universal systems that have a small number of rules. A first result was obtained in [11] where a universal splicing P system with 8 rules was shown. Similar investigations for P systems with symbol-objects were done in [3,1] and the latter article constructs a universal antiport P system with 23 rules.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 95–102, 2010.

In this article we provide a new construction for splicing P systems and prove the remarkable fact that 6 splicing rules are powerful enough for the universality.

2 Definitions

We do not present here definitions concerning the concepts of the theory of formal languages. We refer to [6] and [12] for more details. We only remark that we denote the empty word by ε.

A *tag system* of degree $m > 0$, see [2] and [7], is the triplet $T = (m, V, P)$, where $V = \{a_1, \ldots, a_{n+1}\}$ is an alphabet and where P is a set of productions of form $a_i \to P_i, 1 \leq i \leq n$, $P_i \in V^*$. We remark that for every a_i, $1 \leq i \leq n$, there is exactly one production in P. The symbol a_{n+1} is called the halting symbol. A configuration of the system T is a word w. One passes from a configuration $w = a_{i_1} \ldots a_{i_m} w'$ to the next configuration z by erasing the first m symbols of w and by adding P_{i_1} to the end of the word: $w \Rightarrow z$, if $z = w' P_{i_1}$.

The computation of T over the word $x \in V^*$ is a sequence of configurations $x \Rightarrow \ldots \Rightarrow y$, where either $y = a_{n+1} a_{i_1} \ldots a_{i_{m-1}} y'$, or $y' = y$ and $|y'| < m$. In this case we say that T halts on x and that y is the result of the computation of T over x. We say that T recognizes the language L if there exists a coding φ such that for all $x \in L$, T halts on $\varphi(x)$, and T halts only on words from $\varphi(L)$.

We note that tag systems of degree 2 are able to recognize the family of recursively enumerable languages. Moreover, systems constructed in [2] and [7] have non-empty productions and halt only by reaching the symbol a_{n+1} in the first position.

2.1 Splicing Operations

A *splicing rule* (over an alphabet V) is a 4-tuple (u_1, u_2, u_3, u_4) where $u_1, u_2, u_3, u_4 \in V^*$. It is frequently written as $u_1 \# u_2 \$ u_3 \# u_4$, $\{\$, \#\} \notin V$ or in two dimensions: $\begin{array}{c|c} u_1 & u_2 \\ \hline u_3 & u_4 \end{array}$. Strings $u_1 u_2$ and $u_3 u_4$ are called splicing *sites*.

We say that a word x *matches* rule r if x contains an occurrence of one of the two sites of r. We also say that x and y are *complementary* with respect to a rule r if x contains one site of r and y contains the other one. In this case we also say that x or y may *enter* rule r. When x and y can enter a rule $r = u_1 \# u_2 \$ u_3 \# u_4$, *i.e.*, one has $x = x_1 u_1 u_2 x_2$ and $y = y_1 u_3 u_4 y_2$, it is possible to define the application of r to the couple x, y. The result of this application is w and z where $w = x_1 u_1 u_4 y_2$ and $z = y_1 u_3 u_2 x_2$. We also say that x and y are spliced and w and z are the result of this splicing. We write this as follows: $(x,\ y) \vdash_r (w,\ z)$.

The pair $\sigma = (V, R)$ where V is an alphabet and R is a set of splicing rules is called a *splicing scheme* or an H-scheme.

For a splicing scheme $\sigma = (V, R)$ and for a language $L \subseteq V^*$ we define:

$\sigma(L) \stackrel{\text{def}}{=} \{w, z \in V^* \mid \exists x, y \in L, \exists r \in R : (x,\ y) \vdash_r (w,\ z)\}$.

We now can introduce the iteration of the splicing operation.

$\sigma^0(L) = L$,
$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0$,
$\sigma^*(L) = \cup_{i \geq 0} \sigma^i(L)$.
The iterated splicing preserves the regularity of a language:

Theorem 1. *[9] Let $L \subseteq T^*$ be a regular language and let $\sigma = (T, R)$ be a splicing scheme. Then language $\sigma^*(L)$ is regular.*

An *extended H system* is a quadruple $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$, $A \subseteq V^*$, and R is a set of splicing rules over V. We call V the alphabet of γ, T is the *terminal* alphabet, A is the set of *axioms*. The *language generated* by γ is defined by $L(\gamma) = \sigma^*(A) \cap T^*$, where $\sigma = (V, R)$ is the underlying H scheme of γ.

We say that $\gamma = (V, T, A, R)$ *computes* $L \subseteq V^*$ on input w if $L = L(\gamma')$, where $\gamma' = (V, T, A \cup \{w\}, R)$.

2.2 Splicing (Tissue) P Systems

A *splicing tissue P system* of degree $m \geq 1$ is a construct

$$\Pi = (V, T, G, A_1, \ldots, A_m, R_1, \ldots, R_m),$$

where V is a finite alphabet, $T \subseteq V$ is the terminal alphabet and G is the underlying directed labeled graph of the system. The graph G has m nodes (cells) numbered from 1 to m. Each node i contains a set of strings (a language) A_i over V. Symbols R_i, $1 \leq i \leq m$ are finite sets of rules (associated to nodes) of the form $(r; tar_1, tar_2)$, where r is a splicing rule: $r = u_1 \# u_2 \$ u_3 \# u_4$ and $tar_1, tar_2 \in \{here, out\} \cup \{go_j \mid 1 \leq j \leq m\}$ are target indicators. We remark that the communication graph G can be deduced from the sets of rules. More precisely, G contains an edge (i, j), iff there is a rule $(r; tar_1, tar_2) \in R_i$ with $tar_k = go_j$, $k \in \{1, 2\}$. If one of tar_k is equal to *here*, then G contains the loop (i, i).

A *configuration* of Π is the m-tuple $(N_1, \ldots, N_m)$, where $N_i \subseteq V^*$. A *transition* between two configurations $(N_1, \ldots, N_m) \Rightarrow (N'_1, \ldots, N'_m)$ is defined as follows. In order to pass from one configuration to another, splicing rules of each node are applied in parallel to all possible words that belong to that node. After that, the result of each splicing is distributed according to target indicators. More exactly, if there are x, y in N_i and $r = (u_1 \# u_2 \$ u_3 \# u_4; tar_1, tar_2)$ in R_i, such that $(x,\ y) \vdash_r (w,\ z)$, then words w and z are sent to nodes indicated by tar_1, respectively tar_2. We write this as follows $(x,\ y) \vdash_r (w,\ z)(tar_1, tar_2)$. If $tar_k = here$, $k = 1, 2$, then the word remains in node i (is added to N'_i); if $tar_k = go_j$, then the word is sent to node j (is added to N'_j); if $tar_k = out$, the word is sent outside of the system.

Since the words are present in an arbitrary number of copies, after the application of rule r in node i, words x and y are still present in the same node.

A *computation* in a splicing tissue P system Π is a sequence of transitions between configurations of Π which starts from the initial configuration

$(A_1, \ldots, A_m)$. The result of the computation consists of all words over terminal alphabet T which are sent outside the system at some moment of the computation. We denote by $L(\Pi)$ the language generated by system Π.

We also define the notion of an *input* for the system above. An input word for a system Π is simply a word w over the non-terminal alphabet of Π. The computation of Π on input w is obtained by adding w to the axioms of A_1 and after that by evolving Π as usual. We denote by $L(\Pi, w)$ the result of the computation of Π on w.

We consider the following restricted variant of splicing tissue P systems. A *restricted splicing tissue P system* is a subclass of splicing tissue P systems which has the property that for any rule $(r; tar_1, tar_2)$ either $tar_1 = tar_2 = go_j$, or $tar_1 = tar_2 = out$ or $tar_1 = tar_2 = here$. This means that both resulting strings are moved over the same connection. In this case, we may associate splicing rules to corresponding edges. If both targets are *out*, then we can associate the splicing rule with an edge going to the special node called *out*.

3 Universal Restricted Splicing Tissue P System of Small Size

The universality proofs are based on a simulation of tag systems [2,7] using the well known rotate-and-simulate method. We show that the functioning of any tag system can be simulated using restricted splicing tissue P system with 6 rules. The universality of the corresponding system follow from the existence of universal tag systems.

Let $V = \{a_1, \ldots, a_{n+1}\}$ be an alphabet and $\alpha, \beta \notin V$. Consider coding morphisms c and $\bar{c}$ defined as follows: $c(a_i) = \alpha^{i+1}\beta$, $\bar{c}(a_i) = \beta\alpha^{i+1}$.

Theorem 2. *Let $TS = (2, V, P)$ be a tag system and $w \in V^*$. Then, there is a restricted splicing tissue P system $\Pi = (V', T, G, A_1, A_2, A_3, R_1, R_2, R_3)$, having 6 rules, which given the word $X\beta\beta c(w)\beta Y$ as input simulates TS on input w,* i.e. *such that:*

1. *for any word w on which TS halts producing the result w', the system Π produces a unique result $X'c(w')Y'$,* i.e., $L(\Pi, w) = \{X'c(w')Y'\}$.
2. *for any word w on which TS does not halt, the system Π computes infinitely without producing a result,* i.e., $L(\Pi, w) = \emptyset$.

Proof. We construct the system Π as follows.

Let $|V| = n + 1$. We use the following alphabets V' and T.

$$V' = \{\alpha, \beta, X, X', Y, Y', Z, Z'\}, \qquad T = \{X', Y', \alpha, \beta\}.$$

The initial languages A_j, $j \in \{1, 2, 3\}$ are given as follows.

$$A_1 = \{Z'c(P_i)\bar{c}(a_i)Y \mid a_i \to P_i \in P, 1 \le i \le n\} \cup \{X\beta Z,\ ZY,\ Z'Y'\},$$
$$A_2 = \{XZ\},$$
$$A_3 = \{XZ,\ X'Z\}.$$

The set of rules R_j, $j \in \{1, 2, 3\}$ are given as follows.

$$R_1 = \{1.1 : (\varepsilon\#\beta Y\$Z'\#\varepsilon; go_3, go_3);\ 1.2 : (\varepsilon\#\alpha Y\$Z\#Y; go_2, go_2);$$
$$1.3 : (X\beta\alpha\#\varepsilon\$X\beta\#Z; here, here)\};$$
$$R_2 = \{2.1 : (X\alpha\#\varepsilon\$X\#Z; go_1, go_1)\};$$
$$R_3 = \{3.1 : (X\beta\beta\#\alpha\alpha\$X\#Z; go_1, go_1);\ 3.2 : (X\beta\beta\#\alpha\beta\$X'\#Z; out, out)\}.$$

The graph G can be deduced from the rules above and it is represented in Figure 1. We observe that this graph is induced by a tree structure with duplex communication and loops.

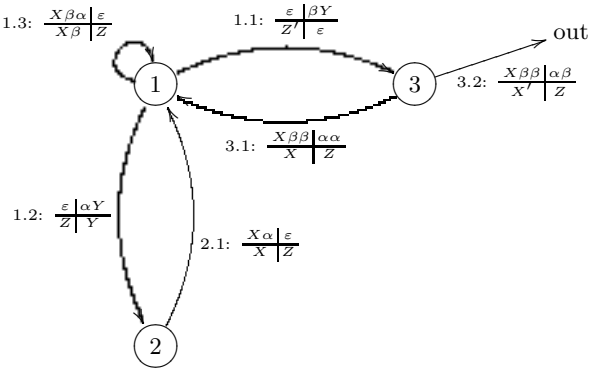


Fig. 1. The communication graph G associated to the construction from Theorem 2

The simulation of TS is performed as follows. For every step of the derivation in TS there is a sequence of several derivation steps in Π. The current configuration w of TS is encoded by a string $X\beta\beta c(w)\beta Y$ present in node 1 of Π (the initial configuration of Π satisfies this property). The simulation of a production $a_i \to P_i$, $1 \leq i \leq n$ is performed using the rotate-and-simulate method used for many proofs in the area of splicing systems. We use this method as follows. First, suffixes $c(P_j)\bar{c}(a_j)$, $1 \leq j \leq n$ are attached to the string producing $X\alpha^i\beta c(a_k w')c(P_j)\beta\alpha^j Y$. After that one symbol α is removed at the left end of the string (rule 2.1) and one symbol α is removed at the right end of the string (rule 1.2). A process of deleting of symbols α continues in the same manner. Hence, only the string for which $j = i$ will remain at the end, producing $X\beta c(a_k w')\beta Y$. After that the symbol a_k is removed (by removing corresponding α's) and a new round begins. The simulation stops when the first symbol is a_{n+1}.

Now we consider a simulation of one step of the derivation in TS. Notice, that in our construction every string that cannot evolve during one step of the simulation cannot evolve anymore. Let $X\beta\beta c(w)\beta Y$ be present in node 1 and $1 \leq j \leq n$. Then, there are two cases with respect to the first letter of w.

1. If $w = a_i a_k w'$, with $1 \leq i \leq n$ and $1 \leq k \leq n+1$ (hence the corresponding configuration in Π will be $X\beta\beta\alpha^i\beta\alpha^k\beta c(w')\beta Y$), then the only possibility is to use the rule 1.1:

$$(X\beta\beta\alpha^i\beta\alpha^k\beta c(w')\beta Y,\ \ Z'c(P_j)\beta\alpha^j Y) \vdash_{1.1} (X\beta\beta\alpha^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^j Y,\ Z'\beta Y),$$

or

$$(X\beta\beta\alpha^i\beta\alpha^k\beta c(w')\beta Y,\ Z'Y') \vdash_{1.1} (X\beta\beta\alpha^i\beta\alpha^k\beta c(w')Y',\ Z'\beta Y).$$

The string $Z'\beta Y$ cannot evolve anymore. The other strings can be used in rule 3.1:

$$(X\beta\beta\alpha^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^j Y,\ XZ) \vdash_{3.1} (Xa^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^j Y,\ X\beta\beta Z),$$
$$(X\beta\beta\alpha^i\beta\alpha^k\beta c(w')Y',\ XZ) \vdash_{3.1} (X\alpha^i\beta\alpha^k\beta c(w')Y',\ X\beta\beta Z).$$

The strings $X\beta\beta Z$ and $X\alpha^i\beta\alpha^k\beta c(w')Y'$ cannot evolve anymore. The remaining strings are of the form $X\alpha^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^j Y$. There are 4 cases with respect to values of i and j.

Case $\mathbf{i > 0}$, $\mathbf{j > 0}$. Then only rule 1.2 is applicable, followed by the application of rule 2.1:

$$(Xa^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^j Y,\ ZY) \vdash_{1.2} (Xa^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^{j-1} Y,\ Z\alpha Y),$$
$$(Xa^i\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^{j-1} Y,\ XZ) \vdash_{2.1} (Xa^{i-1}\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^{j-1} Y,\ X\alpha Z).$$

Hence indices i and j are decremented simultaneously. The strings $Z\alpha Y$ and $X\alpha Z$ cannot evolve anymore.

Case $\mathbf{i > 0}$, $\mathbf{j = 0}$. Then rule 1.1 is applicable:

$$(X\alpha^i\beta\alpha^k\beta c(w')c(P_j)\beta Y,\ Z'c(P_t)\beta\alpha^t Y) \vdash_{1.1}$$
$$(X\alpha^i\beta\alpha^k\beta c(w')c(P_j)c(P_t)\beta\alpha^t Y,\ Z'\beta Y),$$
$$(X\alpha^i\beta\alpha^k\beta c(w')c(P_j)\beta Y,\ Z'Y') \vdash_{1.1} (X\alpha^i\beta\alpha^k\beta c(w')c(P_j)Y',\ Z'\beta Y).$$

All the strings cannot evolve anymore in this case.

Case $\mathbf{i = 0}$, $\mathbf{j > 0}$. Then rule 1.3 can be applied followed by an application of rule 1.2:

$$(X\beta\alpha^k\beta c(w')c(P_j)\beta\alpha^j Y,\ X\beta Z) \vdash_{1.3} (X\beta\alpha^{k-1}\beta c(w')c(P_j)\beta\alpha^j Y,\ X\beta\alpha Z),$$
$$(X\alpha^t\beta c(w')c(P_j)\beta\alpha^j Y,\ ZY) \vdash_{1.2} (X\alpha^t\beta c(w')c(P_j)\beta\alpha^{j-1} Y,\ Z\alpha Y),$$
for some $t \geq 0$.

In this case, the strings cannot evolve anymore.

Case i = 0, j = 0. We observe that in this case we had $i = j$. Then either rule 1.3 can be iteratively applied until $k = 0$, or rule 1.1 is applied when $k > 0$.

$$(X\beta\alpha^k\beta c(w')c(P_i)\beta Y,\ X\beta Z) \vdash_{1.3^*} (X\beta\beta c(w')c(P_i)\beta Y,\ X\beta\alpha Z),$$
$$(X\beta\alpha^k\beta c(w')c(P_i)\beta Y,\ Z'c(P_t)\beta\alpha^t Y) \vdash_{1.1} (X\beta a^k\beta c(w')c(P_i)c(P_t)$$
$$\beta\alpha^t Y,\ Z'\beta Y),$$
$$(X\beta\alpha^k\beta c(w')c(P_i)\beta Y,\ Z'Y') \vdash_{1.1} (X\beta a^k\beta c(w')c(P_i)Y',\ Z'\beta Y).$$

In the first case the string $X\beta\beta c(w'P_i)\beta Y$ is obtained, which corresponds to the string $w'P_i$ of TS. Hence the corresponding production is simulated. In the latter two cases, the resulting strings cannot evolve anymore.

2. If $w = a_{n+1}w'$, (hence the corresponding configuration in Π will be $X\beta\beta\alpha\beta c(w')\beta Y$), then the only possibility is to use rule 1.1:

$$(X\beta\beta\alpha\beta c(w')\beta Y,\ Z'c(P_j)\beta\alpha^j Y) \vdash_{1.1} (X\beta\beta\alpha\beta c(w')c(P_j)\beta\alpha^j Y,\ Z'\beta Y),$$
$$(X\beta\beta\alpha\beta c(w')\beta Y,\ Z'Y') \vdash_{1.1} (X\beta\beta\alpha\beta c(w')Y',\ Z'\beta Y).$$

Then the only applicable rule is rule 3.2.

$$(X\beta\beta\alpha\beta c(w')c(P_j)\beta\alpha^j Y,\ X'Z) \vdash_{3.2} (X'\alpha\beta c(w')c(P_j)\beta\alpha^j Y,\ X\beta\beta Z),$$
$$(X\beta\beta\alpha\beta c(w')Y',\ X'Z) \vdash_{3.2} (X'c(w')Y',\ X\beta\beta Z).$$

Only the string $X'c(w')Y'$ is considered as a result because other strings contain nonterminal symbols.

So, Π correctly simulates one step of the derivation in TS, thus Π correctly simulate whole derivation in TS. It is also clear that a successful computation in TS may be reconstructed from a successful computation in Π. This concludes the proof.

4 Conclusions

In this paper we investigated splicing tissue P systems and we constructed a universal splicing P system with 6 rules. This result is quite remarkable, because the usual implementation of the rotation method for splicing systems needs at least 4 rules. An open problem raised by this result is if the above number is minimal. Other open problems concern the minimal number of rules in the case of other variants of splicing systems, like TVDH systems or splicing test tube systems [9].

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

References

1. Alhazov, A., Verlan, S.: Minimization strategies for maximally parallel multiset rewriting systems. Technical Report 862, TUCS Report No. 862 (2008), http://tucs.fi and arXiv:1009.2706v1 [cs.FL] (September 14, 2010), http://arxiv.org/abs/1009.2706
2. Cocke, J., Minsky, M.: Universality of Tag Systems with P=2. Journal of the ACM 11(1), 15–20 (1964)
3. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G., Verlan, S.: Small Computationally Complete Symport/Antiport P systems. Theoretical Computer Science 372(2-3), 152–164 (2007)
4. Frisco, P.: Computing with Cells: Advances in Membrane Computing. Oxford University Press, Oxford (2009)
5. Head, T.: Formal Language Theory and DNA: an Analysis of the Generative Capacity of Specific Recombinant Behaviors. Bulletin of Mathematical Biology 49(6), 737–759 (1987)
6. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (2001)
7. Minsky, M.: Computations: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
8. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
9. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Heidelberg (1998)
10. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
11. Rogozhin, Y., Verlan, S.: On the Rule Complexity of Universal Tissue P Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 356–362. Springer, Heidelberg (2006)
12. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 3. Springer, Heidelberg (1997)

Membrane Systems Working in Generating and Accepting Modes: Expressiveness and Encodings

Roberto Barbuti[1], Andrea Maggiolo-Schettini[1], Paolo Milazzo[1], and Simone Tini[2]

[1] Dipartimento di Informatica, Università di Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy
{barbuti,maggiolo,milazzo}@di.unipi.it
[2] Dipartimento di Informatica e Comunicazione, Università dell'Insubria
Via Mazzini 5, 21100 Varese and Via Carloni 78, 22100 Como, Italy
simone.tini@uninsubria.it

Abstract. Membrane systems can be seen either as generators or as acceptors of multiset languages. In this paper we compare the expressive power of membrane systems working in accepting mode with that of membrane systems working in generating mode. Features like determinism, presence of promoters and of cooperative rules are considered. The comparison between some of the considered classes of membrane systems is carried out by defining encodings of one class into another.

1 Introduction

Membrane systems (P systems) were introduced by Paun in [13] as distributed parallel computing devices inspired by the structure and the functioning of cells. In the extension of [3] the application of rules may be conditioned by the presence of *promoter objects*. A promoter does not participate in the application of rules, and a single promoter may enable the application of several rules and multiple applications of each one of these rules. P systems with promoters have been shown to be universal even when non-cooperative rules are considered [3]. The same holds for P systems without promoters, but with cooperative rules [14].

In universality proofs P systems based on multiset rewriting rules (transition P systems) are often seen as multiset generators. A computation of a P system gives as output a multiset, and the set of all multisets given as output by different computations of a system are taken as the multiset language generated by the system. An alternative view considers P systems as multiset acceptors. Given a multiset as input, the computation of a P system may take to an accepting state. The set of accepted multisets constitutes the multiset language accepted by the P system. In this paper we are interested in comparing the expressive power of P systems working in generating and in accepting modes. In particular, we consider classes of P systems that are discriminated by admitting, or not, promoters, cooperative rules and nondeterminism, and that do not consider other features like membrane dissolution or creation, priorities, and symport/antiport rules.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 103–118, 2010.

In some cases of interest the comparison will be carried out by defining encodings of one class of generator P systems into the corresponding class of acceptor P systems. These encodings will not only be given to prove some results on the expressive power of the considered classes. They will given also as a tool for a easier construction of P systems accepting languages of interest. In fact, for some classes of languages the definition of a P system generating it could be easier than the definition of an acceptor for it (or viceversa). The proposed encodings may allow to obtain from a P system working in the mode that is easier to be designed an equivalent P system working in the other mode.

The universality results of generator P systems are mostly well-known. As regards acceptors, the universality of nondeterministic acceptor P systems with catalysts has been proved in [9] through simulation of register machines, and the universality of deterministic acceptors has been proved in [5] for P systems with membrane creation and dissolution, and in [5, 10] for P systems with symport/antiport rules. Different notions of P systems working in generating and accepting modes have been considered, for instance in [12, 15], from the point of view of complexity. Other variants of P systems working in accepting mode and with symport/antiport rules have been introduced in [4], called P automata, and in [11], called analyzing P systems. Generating and accepting modes have been considered and compared also in the context of regulated rewriting [6–8].

2 Membrane Systems with Promoters

In this section we recall the definition of P systems with promoters and introduce notions of P systems accepting and generating multiset languages.

2.1 Definition

A membrane system (or P system) consists of a *hierarchy of membranes* that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. As usual, we assume membranes to be labeled by natural numbers. Given a set of objects V, a membrane m contains a multiset of *objects* in V, a set of *evolution rules*, and possibly other membranes, called *child* membranes (m is also called the *parent* of its child membranes). A rule in a membrane m consumes objects in m and produces objects according to target indications, specifying the membranes where each object produced by applying the rule is sent. The products of a rule with target indications are denoted with a set of pairs of the following forms:

- $(v, here)$, meaning that the multiset of objects v produced by the rule remain in the same membrane m;
- (v, out), meaning that the multiset of objects v produced by the rule are sent out of m;
- (v, in_l), meaning that the multiset of objects v produced by the rule are sent into the child membrane l.

An evolution rule may have some *promoters* that are objects required to be present in the membrane m in order to enable the application of the rule. We can assume that all evolution rules have the following form:

$$u \rightarrow (v_h, here)(v_o, out)(v_1, in_{l_1}) \dots (v_n, in_{l_n})|_p$$

where u is the multiset of objects consumed by the rule, $\{l_1, \dots, l_n\}$ is a set of membrane labels, $v_h, v_o, v_1, \dots, v_n$ are the objects (grouped in multisets by target) produced by the rule, and p is the multiset of promoters of the rule. Notice that all objects mentioned in the rules are in V, therefore an object may appear in the reactants, in the products, and in the promoters of the rules. The size of the left–hand side u of an evolution rule is called the *radius* of such a rule. If a P system contains rules of radius greater than one, then it is called a *cooperative* system. Otherwise, it is called *non–cooperative*.

Application of evolution rules is done with maximal parallelism: at each evolution step a multiset of instances of evolution rules is chosen non–deterministically such that no other rule can be applied to the system obtained by removing all the objects necessary to apply all the chosen rules. Rule application consists of removing all the reactants of the chosen rules from the system and adding the products of the rules by taking into account the target indications. Promoters are not consumed by the application of the corresponding evolution rule, thus implying that the presence of a single occurrence of a promoter can enable the application of more than one rule in each maximally parallel evolution step.

A P system has a tree-structure in which the skin membrane is the root and the membranes containing no other membranes are the leaves. We assume membranes labels to be unique: they are assigned at the beginning of the evolution by counting the membranes encountered during a breadth-first visit of the tree–structure, with 1 as the label of the skin membrane.

Now, we formally define P systems with promoters.

Definition 1. *A* P system *is a construct* $\Pi = (V, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$ *where:*

- V *is an* alphabet *whose elements are called* objects*;*
- $\mu \subset \mathbb{N} \times \mathbb{N}$ *is a* membrane structure*, such that* $(l_1, l_2) \in \mu$ *denotes that the membrane labeled by* l_2 *is contained in the membrane labeled by* l_1*;*
- w_j *with* $1 \leq j \leq n$ *are strings from* V^* *representing multisets over* V *associated with the membranes* $1, \dots, n$ *of* μ*;*
- R_j *with* $1 \leq j \leq n$ *are finite sets of* evolution rules *associated with the membranes* $1, \dots, n$ *of* μ*.*

In this paper we assume P systems to be closed computational devices, namely we assume that objects cannot be sent out of the skin membrane (i.e.rules sending objects out are not allowed in the skin membrane) and objects cannot be received by the skin membrane from outside.

A *computation* of a P system is a sequence of maximally parallel evolution steps. A computation is *valid* if and only if the sequence of computation steps

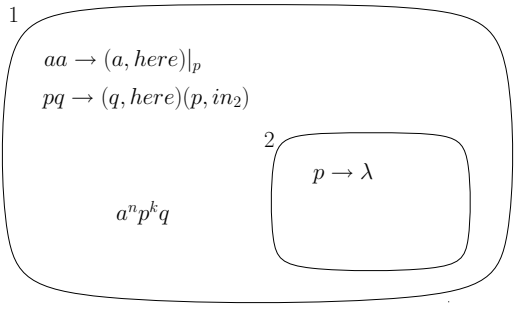


Fig. 1. An example of P system with promoters

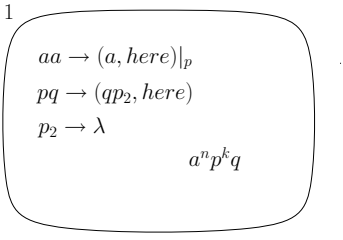


Fig. 2. The result of flattening the P system given in Figure 1

is finite and leads to a *final configuration*, namely a configuration in which no evolution rules can be further applied. A P system is said to be *deterministic* if it may perform only one computation (either valid or not). This happens when at each step there is only one maximal multiset of applicable evolution rules.

We show in Figure 1 an example of P system in which all the main features of the formalism are used. In the figure, membranes are depicted as boxes containing evolution rules, objects and inner membranes. The label of a membrane is at a corner of the corresponding box. Exponents are used to have a compact representation of multiple occurrences of an object in a multiset. For example, $a^n p^k b$ represents the multiset consisting of n occurrences of a, k occurrences of p and one occurrence of b. Symbol λ denotes the empty multiset.

The P system in the figure performs a computation consisting of k steps. At each step the number of a's is halved and one p is sent into membrane 2, where it is canceled. In the final configuration, membrane 1 contains the multiset $a^{\lceil \frac{n}{2^k} \rceil} q$.

Let us assume that V is partitioned into sets Σ and $\mathcal{C}$, where $\mathcal{C}$ is called as the set of *control objects*. From [1] it follows that any P system with promoters Π can be translated into an equivalent P system Π' having a (flat) membrane structure that consists only of the skin. The idea is to obtain the alphabet of the control objects in Π' by enriching the alphabet of the control objects in Π with objects labeled with indexes of membranes in Π to represent objects of Π that are placed in some inner membrane. It turns out that Π and Π' are equivalent in the sense that each of them can mimic the behavior of the other, evolution step by evolution step. At each step the skin membranes of Π and Π' contain the same multisets over Σ. An analogous technique was previously used in [2, 16] but with different classes of P systems. As an example, we show, in Figure 2, the result of flattening the P system with promoters given in Figure 1.

For the sake of precision, the class of P systems considered in [1] is slightly different: (i) promoters of an evolution rule are given as a set rather than a multiset, (ii) membranes are enriched with an interface that filters the objects that can be received from the external environment, and (iii) also rule inhibitors and dissolving rules are considered. As regards (i), it is easy to see that the way in which promoters are given does not influence the flattening technique. As regards (ii), membrane interfaces are used in [1] to ensure compositionality. Here,

we consider closed computational systems, hence we do not allow interactions with the external environment. As regards (iii), it is easy to see that inhibitors and dissolving rules are not introduced by the flattening technique, hence the flat version of a P system of the class we consider here is still in the same class. To sum up, we have that the following result holds.

Theorem 1. *Every P system with promoters can be translated into an equivalent P system whose membrane structure consists only of the skin membrane.*

Proof. Follows from results in [1]. □

If one gives a suitable notion of size of a P system taking into account the cardinality of the set of objects and of the set of rules, one can infer that the flattening technique in [1] takes a P system of size s to a P system of size $o(s)$.

We shall always assume flat P systems in our proofs, consequently we shall always assume the *here* particle for products of evolution rules (e.g. we will write $a \to b$ for $a \to (b, here)$).

In the following we shall study how determinism, promoters, and cooperative rules have an influence on the expressive power of P systems. We shall use the following notations for the different classes of P systems: $P(coo, ndet, pro)$ denotes the class of P systems admitting cooperation, non determinism and promoters, we replace *coo* with *ncoo* in the classes where cooperative rules are not admitted, *ndet* with *det* in the classes where nondeterminism is not admitted, and *pro* with *npro* in the classes where promoters are not admitted.

2.2 Membrane Systems and Multiset Languages

P systems can deal with multiset languages. A multiset language [4] is a set of multisets over a given alphabet. It might be obtained from a language of strings by applying the Parikh mapping to each string in the language. The Parikh mapping takes a string into a vector of natural numbers, in which each element corresponds to the number of occurrences in the string of one of the alphabet symbols. After application of the Parikh mapping, information on the ordering of symbols in the strings of the language is lost.

Among the most important classes of multiset languages we mention the class obtained by applying the Parikh mapping to all context free languages of strings, denoted $PsCF$, that coincides with the class obtained by applying the Parikh mapping to all regular languages, denoted $PsREG$. We mention also the class obtained by applying the Parikh mapping to all recursively enumerable languages of strings, denoted $PsRE$. Note that $PsRE$ is the class of all recursively enumerable sets of multisets (represented as sets of vectors of natural numbers).

A P system can be used either as an acceptor or as a generator of a multiset language over Σ. In the first case, a multiset over Σ is inserted in the skin membrane of the P system and the result of its computations says whether such a multiset belongs to the multiset language accepted by the P system or not. In the second case the P system has a fixed initial configuration and can give as results (possibly in a non-deterministic way) all the possible multisets belonging to a given multiset language.

Let us formalize the notion of P system used as language acceptor.

Definition 2. *A* flat acceptor P system *over an alphabet Σ is a P system $\Pi = (\Sigma \cup \mathcal{C} \cup \{T\}, \emptyset, w_1, R_1)$, where:*

- *$\mathcal{C}$ is a set of* control objects *such that $\Sigma \cap \mathcal{C} = \varnothing$;*
- *T is a special object not contained in $\Sigma \cup \mathcal{C}$;*
- *w_1 is a multiset of objects in $\mathcal{C}$;*
- *A multiset w of objects over Σ is accepted by Π iff when we add w to w_1 then a final configuration can be reached with T appearing in the membrane.*

We remark that one could define equivalent notions of acceptor P systems by assuming that a multiset is accepted if and only if a final configuration can be reached (by ignoring the presence of T). We can simulate this simply by adding T to w_1 and by ensuring that there is no rule in R_1 using such a special object.

We denote the language accepted by a P system Π as $Ps(\Pi)$ (as Parikh set). Moreover, we denote the set of languages accepted by a class of P systems by adding prefix Ps and subscript a in the notation of the class itself as in the following example: we use $PsP_a(ncoo, ndet, pro)$ to denote the set of languages for which there exists an acceptor in the class $P(ncoo, ndet, pro)$.

Now we formalize the notion of P system used as language generator.

Definition 3. *A* flat generator P system *over an alphabet Σ is a P system $\Pi = (\Sigma \cup \mathcal{C}, \emptyset, w_1, R_1)$, where:*

- *$\mathcal{C}$ is a set of* control objects *such that $\Sigma \cap \mathcal{C} = \varnothing$;*
- *w_1 is a multiset of objects in $\mathcal{C}$;*
- *a multiset w of objects over Σ is generated by Π if and only if there exist a multiset $w_o^{\mathcal{C}}$ of objects over $\mathcal{C}$ and a final configuration that can be reached having $w \cup w_o^{\mathcal{C}}$ as multiset of objects.*

Note that also in the case of generator P systems there exist other equivalent definitions that could be considered (and that have been considered in the literature). For instance, one could use a special membrane to collect the output of the system, or could send the output out of the skin membrane.

As for acceptor P systems, we denote the language generated by a P systems Π as $Ps(\Pi)$, and we introduce a notation for the set of languages generated by P systems of a certain class. In this case we replace subscript a with subscript g as in the following example: we use $PsP_g(ncoo, ndet, coo)$ to denote the set of languages for which there exists a generator in the class $P(ncoo, ndet, coo)$.

3 Results on Expressive Power and Encodings

Given a class of P systems $\mathcal{C}$ accepting/generating the class of languages $\mathcal{L}$, and a class of P systems $\mathcal{C}'$ accepting/generating the class of languages $\mathcal{L}'$, we write $\mathcal{L} \Rightarrow \mathcal{L}'$ if we can provide an encoding of $\mathcal{C}$ into $\mathcal{C}'$. Notice that $\mathcal{L} \Rightarrow \mathcal{L}'$ implies $\mathcal{L} \subseteq \mathcal{L}'$, but, in general, if we know that $\mathcal{L} \subseteq \mathcal{L}'$, then giving a mapping from $\mathcal{C}$

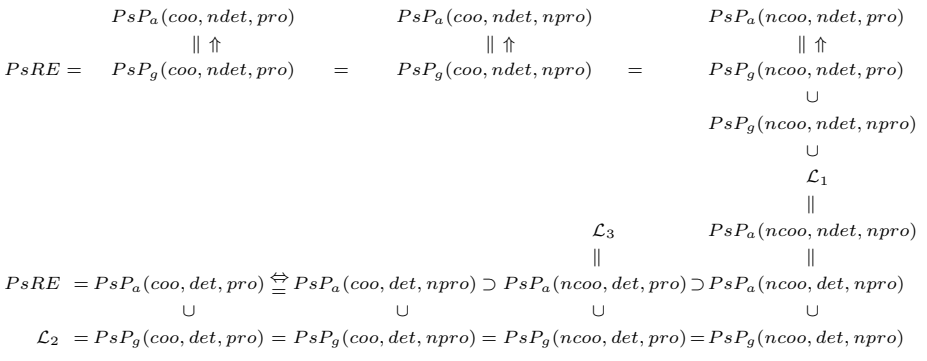


Fig. 3. Relationships among classes of languages accepted and generated by the classes of P systems considered in this paper

to $\mathcal{C}'$ could be not trivial. If we know that both $\mathcal{L} = PsRE$ and $\mathcal{L}' = PsRE$, we could exploit the universal formalism U that has been used to prove $\mathcal{L}' = PsRE$ and give "only" the encoding of $\mathcal{C}$ into U. However, it is often the case that the universal formalism U does not include any form of parallelism. As a result, the encoding of $\mathcal{C}$ (which is based on maximal parallelism) into U is often difficult to be given. Moreover, the loss of parallelism in the encoding of $\mathcal{C}$ into U may cause the acceptor/generator P systems in $\mathcal{C}'$ given by the combination of the two encodings of $\mathcal{C}$ into U and of U into $\mathcal{C}'$, to be much less efficient than the original ones. This motivates the proposal of direct encodings.

In this section we study the expressiveness of several classes of acceptor P systems, by focusing, in particular, on their encoding ability, following the idea that relation $\mathcal{L} \Rightarrow \mathcal{L}'$ is more meaningful than $\mathcal{L} \subseteq \mathcal{L}'$. Our results are summarized in Figure 3, where we report also results that are well known in the literature. Classes $\mathcal{L}_1, \mathcal{L}_2$ and $\mathcal{L}_3$ will be defined later.

Our first three theorems (Thms. 2, 3, 4) show that $PsP_g(ncoo, ndet, pro) \Rightarrow PsP_a(ncoo, ndet, pro)$, $PsP_g(coo, ndet, pro) \Rightarrow PsP_a(coo, ndet, pro)$, and, finally, $PsP_g(coo, ndet, npro) \Rightarrow PsP_a(coo, ndet, npro)$. Since the three classes of generators are universal (see, for instance, [3] and [14]), this implies the universality of these three classes of acceptors. Notice, however, that the universality of the last class was already proved in [9].

If one gives a suitable notion of *size* of a P system taking into account the cardinality of the set of objects and of the set of rules, one can infer that these encodings take a generator of size s to an acceptor of size $o(s)$. Formal proofs of these results are omitted here for lack of space.

In the following, given a multiset of objects v, we shall write v' to denote the multiset $\{a' \mid a \in v\}$, v'' to denote the multiset $\{a'' \mid a \in v\}$, and so on.

Theorem 2. $PsP_g(ncoo, ndet, pro) \Rightarrow PsP_a(ncoo, ndet, pro)$.

Proof. Given any generator Π in $P(ncoo, ndet, pro)$, we construct an equivalent acceptor Π_a in $P(ncoo, ndet, pro)$. By Thm. 1 we can assume that Π is flat. Let

$\Pi = (\Sigma \cup C, \emptyset, w, R)$. Also Π_a will be flat, of the form $(\Sigma_a \cup C_a \cup \{T\}, \emptyset, w_a, R_a)$. The idea is that Π_a embeds Π and, for any input multiset u for Π_a, we exploit Π to generate a multiset v in $Ps(\Pi)$ and, then, we compare u with v: if they coincide then Π_a accepts u. The non-determinism ensures that for every $u \in Ps(\Pi)$ there is an execution by Π_a accepting it.

Actually, when Π is embedded into Π_a, all initial objects in w and all objects mentioned in the rules in R are primed, in order to distinguish them from the input of Π_a, and are considered as control objects. Hence, we have that $\Sigma_a = \Sigma$, $C_a \supseteq \Sigma' \cup C'$ and $w_a \supseteq w'$. Then, in the construction of Π_a we have to face two problems. The first problem is that we must be able to check for termination of Π in order to ensure to compare with the input of Π_a an actual multiset in $Ps(\Pi)$ rather than something that is the result of a partial execution of Π. The second problem is that we must implement the comparison.

To these purposes, let us add two fresh control objects to C_a: s, which triggers the comparison, and g, which is initially in w_a, is produced by all rules derived from Π and prevents the production of s. So, first of all R_a contains all rules:

$$R_\Pi = \{a' \to v'g|_{p'} \mid a \to v|_p \text{ is a rule in } R\}$$

Then, R_a contains the following set of rules, denoted R_1, where also x, x', 1, 2 are fresh objects in C_a and x and 1 are also in w_a:

$$x \to x'|_{1g} \qquad x \to s|_2 \qquad x' \to x|_2 \qquad g \to \lambda \qquad 1 \to 2 \qquad 2 \to 1|_{x'}$$

Sets R_Π and R_1 are such that some instances of g are in Π_a as long as rules in R_Π are applied. When no rule in R_Π can be applied, which simulates the termination of the execution by Π, no occurrence of g is anymore in Π_a. This makes the rule $x \to x'|_{1g}$ no longer applicable. Notice that Π_a either contains both 1 and x, or it contains both 2 and x'. In the former case, since $x \to x'|_{1g}$ cannot be applied and 2 is produced by $1 \to 2$, after one computation step s is produced by $x \to s|_2$ and the comparison is triggered. In the latter case, x and 1 are produced by $x' \to x|_2$ and $2 \to 1|_{x'}$, respectively, and we come back to the previous case. Finally, let us add to C_a also the objects $\overline{0}, \overline{1}, \overline{2}$ and $\overline{3}$ and the set $\hat{C}$ consisting of the capitalized versions of the symbols in $\Sigma \cup \Sigma'$. Moreover, let us add $\overline{0}$ and T to w_a. The comparison of the multiset generated by the rules in R_Π with the input is performed by the following set of rules (denoted R_2):

$$\overline{0} \to \overline{1}|_s \qquad \overline{1} \to \overline{2} \qquad \overline{2} \to \overline{3} \qquad \{\,\overline{3} \to \overline{1}|_{Taa'} \mid a \in \Sigma\}$$

$$\{\,a \to a|_{\overline{1}} \mid a \in \Sigma\,\} \qquad \{\,a \to A|_{\overline{1}} \mid a \in \Sigma\,\}$$
$$\{\,a' \to a'|_{\overline{1}} \mid a \in \Sigma\,\} \qquad \{\,a' \to A'|_{\overline{1}} \mid a \in \Sigma\,\}$$

$$\{\,T \to \lambda|_{AB\overline{2}} \mid A, B \in \hat{C}\,\} \qquad \{\,T \to \lambda|_{A'B'\overline{2}} \mid A', B' \in \hat{C}\,\}$$

$$\{\,T \to \lambda|_{A\overline{3}} \mid A \in \hat{C}\,\} \qquad \{\,T \to \lambda|_{A'\overline{3}} \mid A' \in \hat{C}\,\} \qquad \{\,T \to T|_{AA'\overline{3}} \mid A, A' \in \hat{C}\,\}$$
$$\{\,A \to \lambda|_{\overline{3}} \mid A \in \hat{C}\,\} \qquad \{\,A' \to \lambda|_{\overline{3}} \mid A' \in \hat{C}\,\}$$

Objects $\overline{0}, \overline{1}, \overline{2}$ and $\overline{3}$ are used to sequentialize different phases of the comparison. In particular, the rule consuming $\overline{0}$ starts the comparison, and it is triggered by s when the rules in R_Π are no longer applicable.

Rules promoted by $\overline{1}$ transform a non-deterministically chosen portion of the objects of the input multiset and of the multiset generated by R_Π into their capitalized version. Rules promoted by $\overline{2}$ check that at most one object of each of the two multisets to be compared has been capitalized, otherwise they replace T with λ. Rules promoted by $\overline{3}$ check that at least one object of each of the two multisets to be compared has been capitalized, and that such two objects are one the primed version of the other. Moreover, the capitalized objects are deleted. These three phases remove one object from the input multiset and the corresponding primed one from the multiset generated by R_Π, and they are repeated until there are pairs of corresponding objects in the two multisets and T has not been removed. Such a control is performed by the promoters of rules consuming $\overline{3}$.

Summarizing, $\Sigma_a = \Sigma$, $C_a = \Sigma' \cup C' \cup \hat{C} \cup \{\overline{0}, \overline{1}, \overline{2}, \overline{3}, x, x', 1, 2, g, s\}$, $w_a = w' \cup \{\overline{0}, 1, x, g, T\}$ and $R_a = R_\Pi \cup R_1 \cup R_2$. □

Theorem 3. $PsP_g(coo, ndet, pro) \Rightarrow PsP_a(coo, ndet, pro)$.

Proof. Following the proof of Thm. 2, given any generator Π in $P(coo, ndet, pro)$ we construct an acceptor Π_a in $P(coo, ndet, pro)$ that embeds Π. The acceptor Π_a can be constructed exactly as in the proof of Thm. 2. However, since here we can exploit cooperative rules, the set of rules R_2 used in the proof of Thm. 2 to implement the comparison between the input of Π_a and the multiset generated by R_Π can be replaced by the following rules:

$$\{aa' \to \lambda|_s \mid a \in \Sigma\} \cup \{aT \to \lambda|_s \mid a \in \Sigma\} \cup \{a'T \to \lambda|_s \mid a \in \Sigma\}$$

In this case the comparison requires only one computation step. □

Theorem 4. $PsP_g(coo, ndet, npro) \Rightarrow PsP_a(coo, ndet, npro)$.

Proof. Also in this case we exploit the generator Π in $P(coo, ndet, npro)$ to build an equivalent acceptor Π_a in $P(coo, ndet, npro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C, \emptyset, w, R)$. Also Π_a will be flat, of the form $(\Sigma_a \cup C_a \cup \{T\}, \emptyset, w_a, R_a)$. As in the proof of Thm. 2, we rename all objects in Π so that $\Sigma_a = \Sigma$, $C_a \supseteq \Sigma' \cup C'$, $w_a \supseteq w'$, and we introduce a fresh control object s in C_a triggering the comparison between the input multiset of Π_a and the multiset generated by the rules in Π_a derived from those in Π. Here s is triggered by another control object $t \in C_a$, trough the rule

$$t \to rs$$

where also r is a fresh object in C_a. The idea is that t is initially in w_a and the other rules in R_a will ensure that in all computations leading to a final configuration with T in the membrane, then this rule fires only after the rules derived from those in Π have generated their multiset.

Acceptor Π_a performs a loop with 3 steps, until s is produced from t by rule $t \to rs$. The sequence of these 3 steps simulates a single computation step by Π. At the first step in the loop, the following set of rules R_Π^1 may fire:

$$\{u' \to v''v''', tu' \to v''v'''t' \mid u \to v \text{ is a rule in } R\}$$

Firing $tu' \to v''v'''t'$ prevents firing $t \to rs$ and, as a consequence, the production of s. Object t' is in C_a and serves to produce t once more.

At the second step, the following set of rules R^2_Π may fire:

$$\{a'''rT \to \lambda \mid a \in \Sigma\} \cup \{t' \to t''\} \cup \{a'' \to a'''' \mid a \in \Sigma\}$$

The rules in the first set check that s has not been produced (note that s can be produced only together with r) if the computation by Π has not terminated yet. More precisely, if s has been already produced and the computation by Π has not terminated yet, T is removed.

At the third step the following set of rules R^3_Π may fire:

$$\{a'''a'''' \to a' \mid a \in \Sigma\} \cup \{t'' \to t\}$$

so that the first step can begin once more.

When Π_a exits from the loop, which simulates the termination by Π, s can be exploited for the comparison implemented by the following set of rules R_1:

$$\{saa' \to s \mid a \in \Sigma\} \cup \{saT \to \lambda \mid a \in \Sigma\} \cup \{sa'T \to \lambda \mid a \in \Sigma\}$$

Summarizing, $\Sigma_a = \Sigma$, $C_a = \{a', a'', a''', a'''' \mid a \in \Sigma \cup C\} \cup \{r, s, t, t', t''\}$, $w_a = w' \cup \{t, T\}$, $R_a = \{t \to rs\} \cup R^1_\Pi \cup R^2_\Pi \cup R^3_\Pi \cup R_1$. □

From our first three theorems, the following results follow.

Corollary 1. *It holds that:*

- $PsP_g(ncoo, ndet, pro) \subseteq PsP_a(ncoo, ndet, pro)$,
- $PsP_g(coo, ndet, pro) \subseteq PsP_a(coo, ndet, pro)$,
- $PsP_g(coo, ndet, npro) \subseteq PsP_a(coo, ndet, npro)$.

Let us prove now that if we admit neither promoters nor cooperative rules, then nondeterministic acceptors and deterministic acceptors have the same expressive power, and are less expressive than nondeterministic generators. To this purpose, we characterize both $PsP_a(ncoo, ndet, npro)$ and $PsP_a(ncoo, det, npro)$.

Given a set of objects Σ and $A, N \subseteq \Sigma$, let $L_{A,N}$ and L_N denote the following multiset languages:

$$L_{A,N} = \{u \mid A \cap u \neq \emptyset \text{ and } N \cap u = \emptyset\}, \quad L_N = \{u \mid N \cap u = \emptyset\}.$$

Let $\mathcal{L}_1$ be the class $\mathcal{L}_1 = \{L_{A,N} \mid A, N \subseteq \Sigma \text{ for some set of objects } \Sigma\} \cup \{L_N \mid N \subseteq \Sigma \text{ for some set of objects } \Sigma\}$.

Theorem 5. $PsP_a(ncoo, ndet, npro) = PsP_a(ncoo, det, npro) = \mathcal{L}_1$.

Proof. First of all we prove that $\mathcal{L}_1 \subseteq PsP_a(ncoo, det, npro)$. Given a set of objects Σ and sets $A, N \subseteq \Sigma$, an acceptor for $L_{A,N}$ has no control object and rules $\{a \to T \mid a \in A\}$ and $\{b \to b \mid b \in N\}$. An acceptor for L_N contains initially an occurrence of T and has rules $\{b \to b \mid b \in N\}$.

It remains to prove that $PsP_a(ncoo, ndet, npro) \subseteq \mathcal{L}_1$. Assume any acceptor $\Pi \in P(ncoo, ndet, npro)$. If it contains a rule of the form $T \to u$, for any $u \in \Sigma^*$,

then $Ps(\Pi) = \emptyset$, and $\emptyset \in \mathcal{L}_1$ ($\emptyset = L_\Sigma$). Otherwise, let G be the graph having a node for each object in $\Sigma \cup C$ and an arch from a to b if there is a rule $a \to u$ with $b \in u$. Let N be the set of the objects $a \in \Sigma$ such that all paths from a are infinite, meaning that there exists an object a' such that $a \to \cdots \to a'$ and $a' \to \cdots \to a'$, and let A be the set of the objects $a \in \Sigma$ such that at least one path from a is finite and leads to T, namely has the form $a \to \cdots \to T$. If T is an initial object in Π then a multiset is accepted iff it gives rise to a finite computation, because no rule can remove T and the final configuration, if reached, contains T for sure. Therefore, $Ps(\Pi) = L_N$. If T is not initially in Π, then a multiset is accepted iff it gives rise to a finite computation that introduces T in one of its steps. Therefore, $Ps(\Pi) = L_{A,N}$. □

Corollary 2. $PsP_g(ncoo, ndet, npro) \supset PsP_a(ncoo, ndet, npro)$.

Proof. Follows from results in [14], where P systems without cooperative rules and with the output interpreted as a natural number are proven to be able to generate semilinear set of numbers (Theorem 3.3.2). In the proof of the theorem a translation of context free grammars into P systems without cooperation is given which implies that $PsREG \subseteq PsP_g(ncoo, ndet, npro)$. It is obvious that $\mathcal{L}_1 \subset PsREG$. (Recall that $PsREG = PsCF$.) □

Let us switch to deterministic systems. In this case the expressive power of generators is quite poor, and equivalent to the class of multiset languages consisting of at most one multiset. Let $\mathcal{L}_2 = \{\{w\} \mid w \text{ is a multiset}\} \cup \{\varnothing\}$.

Proposition 1. $PsP_g(ncoo, det, pro) = PsP_g(coo, det, pro) = PsP_g(ncoo, det, npro) = PsP_g(coo, det, npro) = \mathcal{L}_2$.

Proof. The initial configuration of a generator P system is fixed. Determinism implies that there is one only possible execution. If such an execution terminates, then it gives the only multiset of the language as output, otherwise the generated language is empty. Any language consisting of one multiset u can be generated by means of a non-cooperative rule without promoters $a \to v$ at the first step, where a is a control object initially in the P system. Cooperation and promoters do not increase expressiveness. □

We already know that $PsP_a(ncoo, det, npro) = \mathcal{L}_1$. Let us characterize the class $PsP_a(ncoo, det, pro)$, which turns out to be more expressive. Let $\mathcal{L}_3$ denote the least class of multiset languages including all sets $\{a^n | n \geq k\}$ for every object a and every $k \in \mathbb{N}$, closed by complementation, finite union and finite intersection. Let us write $u \xrightarrow{R} v$ to denote that by performing a multiset of rules R we rewrite a multiset of objects u into a multiset of objects v. Let us write $u \models r$ if a multiset of objects u triggers a rule r.

Theorem 6. $PsP_a(ncoo, det, pro) = \mathcal{L}_3$.

Proof. Let us prove first that $PsP_a(ncoo, det, pro) \subseteq \mathcal{L}_3$. Let Π be an acceptor in $P(ncoo, det, pro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C \cup$

$\{T\}, \emptyset, w, R)$. Assume that $R = \{r_i \mid i \in I\}$, with each r_i of the form $a_i \to u_i|_{p_i}$. Let $m = \max\{h \mid \exists i \in I, b \in \Sigma.\, b^h \in p_i\}$.

To prove that $Ps(\Pi) \in \mathcal{L}_3$ it is enough to prove that if $a^h u \in Ps(\Pi)$ for some $a \in \Sigma$, $u \in \Sigma^*$ and $h \geq m$, then also $a^{h+1}u \in Ps(\Pi)$.

Assume that $a^h u$ is accepted by Π by performing n computation steps, for some $n \in \mathbb{N}$. More precisely, there exist n multisets $T_1, \ldots, T_n$, with $T_j = \{r_i^{n_{i,j}} \mid i \in I\}$, and $n+1$ multisets of objects $u_0, u_1, \ldots, u_n$ such that $u_0 \stackrel{T_1}{\to} u_1 \stackrel{T_2}{\to} \cdots \stackrel{T_n}{\to} u_n$, $u_0 = a^h u$, $T \in u_n$ and $u_n \not\models r_i$ for any $i \in I$.

Let $v_0 = a^{h+1}u$. We can prove that there exist n multisets $T'_1, \ldots, T'_n$, with $T'_j = \{r_i^{n'_{i,j}} \mid i \in I\}$, and n multisets of objects $v_1, \ldots, v_n$ such that $v_0 \stackrel{T'_1}{\to} v_1 \stackrel{T'_2}{\to} \cdots \stackrel{T'_n}{\to} v_n$, $v_n \not\models r_i$ for any $i \in I$, $n_{i,j} \leq n'_{i,j}$ and $n_{i,j} = 0 \Rightarrow n'_{i,j} = 0$ for each $i \in I$ and $1 \leq j \leq n$, and, finally, $v_j \supseteq u_j$ and $b^x \in u_j$ and $b^{x+y} \in v_j$ with $y > 0$ and $b^{x+y} \not\subseteq u_j$ imply $x \geq m$ for each $0 \leq j \leq n$.

In fact, we know that $v_0 \supseteq u_0$ and that $b^x \in u_0$ and $b^{x+y} \in v_0$ with $y > 0$ and $b^{x+y} \not\subseteq u_0$ imply that b is a, and, therefore, $x = h \geq m$. Now, given any $0 \leq j \leq n$, assume that $v_j \supseteq u_j$ and that $b^x \in u_j$ and $b^{x+y} \in v_j$ with $y > 0$ and $b^{x+y} \not\subseteq u_j$ imply $x \geq m$. This implies that v_j and u_j promote the same rules. Therefore, if $j = n$, since $u_j \not\models r_i$ for any $i \in I$, we infer that also $v_j \not\models r_i$ for any $i \in I$. If $j < n$ then $v_j \supseteq u_j$ implies that a computation step $v_j \stackrel{T'_j}{\to} v_{j+1}$ with $n'_{i,j} \geq n_{i,j}$ actually exists. Since u_j and v_j promote the same rules and all rules are non-cooperative, we also infer that $n_{i,j} = 0$ implies $n'_{i,j} = 0$. Moreover, $u_{j+1} = (u_j \cap \{b \mid b \neq a_i \text{ for any } i \text{ with } n_{i,j} > 0\}) \cup \{u_i^{n_{i,j}} \mid i \in I\}$ and $v_{j+1} = (v_j \cap \{b \mid b \neq a_i \text{ for any } i \text{ with } n_{i,j} > 0\}) \cup \{u_i^{n'_{i,j}} \mid i \in I\}$. The relation $u_{j+1} \subseteq v_{j+1}$ follows immediately. It remains to prove that it cannot happen that $b^x \in u_{j+1}$, $b^{x+y} \in v_{j+1}$, $y > 0$, $b^{x+y} \not\subseteq u_{j+1}$ and $x < m$ for any $b \in \Sigma$. If, by contradiction, this happens for some b, then there is a rule $r_i = a_i \to u_i|_{p_i}$ with $b \in u_i$ such that $n'_{i,j} > n_{i,j}$ and $n_{i,j} < x + y$. We infer that $a_i^{n'_{i,j}} \in v_j$ and $a_i^{n_{i,j}} \in u_j$. Since we know that for all c it holds that $c^k \in u_j$ and $c^{k+h} \in v_j$ with $h > 0$ imply $k \geq m$, we infer that $n_{i,j} \geq m$. Having $n_{i,j} \geq m$ and $x < m$ is a contradiction, since $x \geq n_{i,j}$.

Let us prove now that $PsP_a(ncoo, det, pro) \supseteq \mathcal{L}_3$. Actually, we prove that each multiset in $\mathcal{L}_3$ is accepted by a P system in $P(ncoo, det, pro)$ that always terminates and that consumes neither objects in Σ nor the symbol T.

The multiset $\{a^n \mid n \geq k\}$ is accepted by a P system with a control object b, initial multiset b and a rule $b \to T|_{a^k}$.

The multiset $\{a^n \mid n < k\}$ is accepted by a P system with control objects $b, 1, 2$, initial multiset $b1$ and rules $b \to \lambda|_{a^k}$, $1 \to 2$ and $2 \to T|_b$.

Finally, assume two multiset languages L_1 and L_2 in $\mathcal{L}_3$ and let Π_1 and Π_2 be the P systems accepting them. Assume that the set of control objects in Π_1 and Π_2 are disjoint. Let Π'_1 and Π'_2 be the P systems obtained from Π_1 and Π_2 by replacing T with T_1 and T with T_2, respectively. The language $L_1 \cap L_2$ is built by joining all control objects and rules in Π'_1 with all control objects and rules in Π'_2 and by adding the rule $T_1 \to T|_{T_2}$. The language $L_1 \cup L_2$ is built

by joining all objects and rules in Π'_1 with all objects and rules in Π'_2 and by adding the rules $T_1 \to T$ and $T_2 \to T$. □

Let us prove now the universality of the class $PsP_a(coo, det, npro)$. In this case the corresponding class $PsP_g(coo, det, npro)$ is less expressive, therefore looking for an encoding of $PsP_g(coo, det, npro)$ into $PsP_a(coo, det, npro)$ as done in Thms. 2, 3, 4 is useless. We directly simulate 3-register machines, for which universality has been proven without the need of any complicated representation of data and instructions (as it happens with 2-register machines). Notice that the universality of deterministic acceptors was already proved in [5] and [10], for P systems with symport and antiport rules.

Theorem 7. $PsP_a(coo, det, npro) = PsRE$.

Proof. We provide a map assigning to a 3-register machine M an equivalent acceptor P system Π_M in $P(coo, det, npro)$. Let R_1, R_2 and R_3 be the three registers of M, and $0 \leq i \leq m$ be the labels of its instructions. A state of M is a triple (i, A, B, C), with $0 \leq i \leq m$ and $A, B, C \in \mathbb{N}$, the initial state is $(1, A', B', C')$ for some $A', B', C' \in \mathbb{N}$, and the pair (A', B', C') is accepted if M starting from $(1, A', B', C')$ reaches $(0, 0, 0, 0)$. The idea is that Π_M uses objects i with $0 \leq i \leq n$, a, b and c, and represents a configuration (i, A, B, C) with multiset $(ia^A b^B c^C)$.

Instruction $i : R_1+, j$ is simulated by rule $i \to aj$.

Instruction $i : R_1-, j, k$ is simulated by rules

$$i \to x_i y_i \qquad a x_i \to x'_i \qquad y_i \to y'_i \qquad y'_i x'_i \to j \qquad y'_i x_i \to k\,.$$

Instructions over R_2 and R_3 are analogous, we simply replace any occurrence of a with b or c, respectively. Finally, we need the following rules:

$$0 \to T \qquad Ta \to \lambda \qquad Tb \to \lambda \qquad Tc \to \lambda\,.$$ □

All results over deterministic acceptors proved so far can be summarized in the following corollary.

Corollary 3. $PsP_a(ncoo, det, npro) \subset PsP_a(ncoo, det, pro) \subset PsP_a(coo, det, npro) = PsP_a(coo, det, pro)$.

Proof. Directly from Thm. 5, Thm. 6 and Thm. 7 □

Moreover, we have that each class of deterministic generators is strictly included in the corresponding class of acceptors.

Corollary 4. *It holds that:*

- $PsP_g(ncoo, det, npro) \subset PsP_a(ncoo, det, npro)$;
- $PsP_g(ncoo, det, pro) \subset PsP_a(ncoo, det, pro)$;
- $PsP_g(coo, det, npro) \subset PsP_a(coo, det, npro)$;
- $PsP_g(coo, det, pro) \subset PsP_a(coo, det, pro)$.

Proof. Directly from Thm. 5, Prop. 1, Cor. 3 and $\mathcal{L}_2 \subset \mathcal{L}_1$.

Finally, we provide an encoding of deterministic acceptors with cooperative rules and promoters into the subclass without promoters.

Theorem 8. $PsP_a(coo, det, pro) \Leftrightarrow PsP_a(coo, det, npro)$.

Proof. The encoding $PsP_a(coo, det, npro) \Rightarrow PsP_a(coo, det, pro)$ is obvious. As regards the other direction, given any acceptor $\Pi \in P(coo, det, pro)$, we derive an equivalent acceptor $\hat{\Pi} \in P(coo, det, npro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C, \emptyset, w, R)$. Assume that $R = \{r_1, \ldots, r_k\}$ and $\Sigma \cup C = \{a_1, \ldots, a_n\}$. Also $\hat{\Pi}$ will be flat, of the form $\hat{\Pi} = (\hat{\Sigma} \cup \hat{C}, \emptyset, \hat{w}, \hat{R})$, with $\hat{\Sigma} = \Sigma$, $\hat{C} \supset C$ and $\hat{w} \supseteq w$.

The starting idea is that for any rule $r_i \equiv u_i \to v_i|_{p_i}$ in R we have a rule r'_i without promoters in $\hat{R}$ of the form $r'_i \equiv u_i p_i \to v_i p_i$, so that performing a step S by Π with n_i occurrences of r_i in parallel for each $1 \leq i \leq k$ is simulated by performing n_i occurrences of r'_i in sequence, for each $1 \leq i \leq k$.

Rules r_i as above do not work, for three reasons. The first point is that if $u_i \cap p_i \neq \emptyset$ then r_i is triggered by $(u_i \cup p_i) \setminus (u_i \cap p_i)$, whereas r'_i requires the whole multiset $u_i \cup p_i$. This can be repaired by rewriting all r_i as $u_i \to v_i|_{p_i \setminus u_i}$, without modifying the behavior of Π, before deriving $\hat{\Pi}$ from Π. The second point is that by moving promoters to left hand sides of rules we may introduce nondeterminism. (For example, by transforming rules $a \to d|_c$ and $b \to e|_c$ into $ac \to dc$ and $bc \to ec$.) This can be repaired by rewriting r'_i as $iu_i p_i \to v_i p_i$, where $1 \leq i \leq k$ are new control objects in $\hat{C}$ that must be introduced in sequence. The third point is that if $v_i \cap u_i \neq \emptyset$ then performing r'_i may trigger r'_i itself, which should be prevented when we are simulating a single evolution step S by Π. This can be repaired by rewriting r'_i as $iu'_i p_i \to v''_i p_i$, provided that new control objects a', a'' are introduced in $\hat{C}$ for each $a \in \Sigma \cup C$, objects a are rewritten into a' before the sequence of steps by $\hat{\Pi}$ simulating S and objects a'' are rewritten into a after the same sequence.

The initial multiset $\hat{w}$ contains two fresh control objects $s, s' \in \hat{C}$. Acceptor $\hat{\Pi}$ may start by performing the following rules:

$$ss' \to 1s'\hat{1}\overline{1} \qquad R_a = \{a \to a' \mid a \in \Sigma \cup C\}$$

so that the object 1 triggering r'_1 is introduced and all objects in $\Sigma \cup C$ are primed. Also s', $\hat{1}$ and $\overline{1}$ are new control objects in $\hat{C}$, whose role will be clarified later.

Then, for each $1 \leq i \leq k$, $\hat{C}$ contains also objects i', i'', i''', i'''', and r'_i is adjusted once more as

$$r'_i \equiv i' u'_i p'_i \to v''_i p'_i i'''$$

For each $1 \leq i < k$ the following set of rules are added to $\hat{R}$

$$R_i = \{i \to i'i'', i'' \to i''''', i'''i'''' \to i, i'i'''' \to i + 1\}$$

so that i is rewritten into $i + 1$ as soon as r'_i remains without $u'_i p'_i$. Moreover, the following rules are added to $\hat{R}$

$$R_k = \{k \to k'k'', k'' \to k''''', k'''k'''' \to k, k'k'''' \to v_1\}$$

where the role of the control object $v_1 \in \hat{C}$ will be explained later.

Notice that a step S by Π consisting of n_i occurrences of r_i for each $1 \leq i \leq n$ is simulated by performing n_1 occurrences of r'_1 in sequence, then n_2 occurrences of r'_2 in sequence and so on. This is correct only if Π is deterministic, as in our case. In fact, when Π is nondeterministic, our strategy solves the nondeterminism and, therefore, does not simulate some valid behaviors by Π.

It remains to map any a' that has not been consumed by $r'_1, \ldots, r'_k$ and any a'' that has been introduced by $r'_1, \ldots, r'_k$ to a. To this purpose, first of all we add to $\hat{C}$ new control objects $v_1, \ldots, v_n$ and $t_1, \ldots, t_n$, respectively.

Then, for each $1 \leq j \leq n$, $\hat{R}$ contains the following set of evolution rules

$$R^j = \{v_j \to v'_j v''_j, v'_j a'_j \to v'''_j a_j, v''_j \to v''''_j, v'''_j v''''_j \to v_j, v'_j v''''_j \to t_j\}$$

mapping all a'_js to a_j, where also $v'_j, v''_j, v'''_j, v''''_j$ are new objects in $\hat{C}$. When this task has been completed, t_j is introduced. Moreover, for all $1 \leq j < n$, $\hat{R}$ contains the following set of evolution rules

$$R^j_* = \{t_j \to t'_j t''_j, t'_j a''_j \to t'''_j a_j, t''_j \to t''''_j, t'''_j t''''_j \to t_j, t'_j t''''_j \to v_{j+1}\}$$

mapping all a''_js to a_j, where also $t'_j, t''_j, t'''_j, t''''_j$ are new objects in $\hat{C}$. Finally, $\hat{R}$ contains also the following set of evolution rules

$$R^n_* = \{t_n \to t'_n t''_n, t'_n a''_n \to t'''_n a_n, t''_n \to t''''_n, t'''_n t''''_n \to t_n, t'_n t''''_n \to t\}$$

mapping all a''_ns to a_n. When this task has been completed, object $t \in \hat{C}$ is introduced, which is exploited to trigger the rule $t \to s$ so that s and s' can reintroduce 1 once more and the subsequent step by Π can be simulated. It remains to solve a point: When no rule $r'_1, \ldots, r'_k$ is triggered, $\hat{\Pi}$ does not terminate but enters an infinite loop. Notice that this happens when object v_1 is introduced in $3k$ steps after object 1. So, let us rewrite the rule $k'k'''' \to v_1$ introducing v_1 as $k'k'''' \to v_1 z z'$, where z, z' are new control objects in $\hat{C}$, and let us add to $\hat{R}$ the following set of rules R':

$$\{\widehat{i} \to \widehat{i+1} \mid 1 \leq i < 3k\} \qquad \{\overline{i} \to \overline{i+1} \mid 1 \leq i \leq 3k\}$$
$$z\widehat{3k}s' \to \lambda \qquad \overline{3k+1}\widehat{3k} \to \lambda \qquad z' \to z'' \qquad zz'' \to \lambda$$

so that when $\widehat{3k}$ and z appear at the same step then s' is removed and the infinite loop is prevented. Objects $\overline{i}$ are needed since $\overline{3k+1}$ removes $\widehat{3k}$ when $\widehat{3k}$ appears before z. □

4 Conclusions

The paper has studied relationships among classes of multiset languages accepted and generated by P systems. In particular, the role of determinism, presence of promoters and cooperative rules have been considered.

In the nondeterministic case, when either promoters or cooperative rules are allowed, acceptor P systems have shown to be universal. The same is known to hold for the corresponding classes of nondeterministic generator P systems. In

the deterministic case, acceptor P systems have been shown to be universal only if cooperative rules are allowed. Universality has been shown not to hold for the corresponding classes of generator P systems.

All the considered classes of languages have been characterized, and results of strict inclusion among some of them have been proved. Moreover, in some cases, we have been able to give encodings of P systems working in generating mode into the corresponding P systems working in accepting mode. The definition of other encodings between the considered classes of P systems and the study of the complexity aspects of these encodings (both in terms of the complexity of the encodings themselves and of the size and efficiency of the results of the encoding with respect to the corresponding input) will be the subject of further work.

References

1. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: A P Systems Flat Form Preserving Step-by-step Behaviour. Fundamenta Informaticae 87, 1–34 (2008)
2. Bianco, L., Manca, V.: Encoding–Decoding Transitional Systems for Classes of P Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 134–143. Springer, Heidelberg (2006)
3. Bottoni, P., Martin-Víde, C., Păun, G., Rozenberg, G.: Membrane Systems with Promoters/Inhibitors. Acta Informatica 38, 695–720 (2002)
4. Csuhaj-Varjú, E.: P Automata. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 19–35. Springer, Heidelberg (2005)
5. Calude, C.S., Păun, G.: Bio-steps beyond Turing. BioSystems 77, 175–194 (2004)
6. Fernau, H.: Graph-controlled Grammars as Language Acceptors. Journal of Automata, Languages and Combinatorics 2, 79–91 (1997)
7. Fernau, H., Holzer, M.: Accepting Multi-Agent Systems II. Acta Cybernetica 12, 361–380 (1996)
8. Fernau, H., Holzer, M., Bordihn, H.: Accepting Multi-Agent Systems. Computers and Artificial Intelligence 15, 123–139 (1996)
9. Freund, R., Kari, L., Oswald, M., Sosk, P.: Computationally Universal P Systems Without Priorities: Two Catalysts Are Sufficient. Theor. Comput. Sci. 330(2), 251–266 (2005)
10. Freund, O., Păun, G.: On Deterministic P Systems (Manuscript), http://ppage.psystems.eu/
11. Freund, R., Oswald, M.: A short note on analysing P systems. Bulletin of the EATCS 79, 231–236 (2002)
12. Ibarra, O.H.: On the Computational Complexity of Membrane Systems. Theoretical Computer Science 320, 89–109 (2004)
13. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 61, 108–143 (2000)
14. Păun, G.: Membrane computing. An introduction. Springer, Berlin (2002)
15. Porreca, A.E., Mauri, G., Zandron, C.: Complexity Classes for Membrane Systems. Theoretical Informatics and Applications 40, 141–162 (2006)
16. Qi, Z., You, J., Mao, H.: P systems and Petri nets. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 286–303. Springer, Heidelberg (2004)

BioSimWare: A Software for the Modeling, Simulation and Analysis of Biological Systems

Daniela Besozzi[1], Paolo Cazzaniga[2], Giancarlo Mauri[2], and Dario Pescini[2]

[1] Università degli Studi di Milano
Dipartimento di Informatica e Comunicazione
Via Comelico 39, 20135 Milano, Italy
besozzi@dico.unimi.it
[2] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
{cazzaniga,mauri,pescini}@disco.unimib.it

Abstract. BioSimWare is a novel software that provides a user-friendly framework for the modeling and stochastic simulation of complex biological systems, ranging from cellular processes to population phenomena. BioSimWare implements several stochastic algorithms to simulate the dynamics of single or multi-volume models, as well as automatic tools to analyze the effect of variation of the system parameters. BioSimWare supports SBML format, and can automatically convert stochastic models into the corresponding deterministic formulation. The main features of BioSimWare are presented in this paper, together with some applications which highlight the most relevant aspects of the computational tools that it provides.

1 Introduction

Over the last decades, multidisciplinary approaches aimed at the analysis of cellular systems have aroused an increasing attention of both experimental and theoretical researchers, for their promising capability of gaining an unprecedented understanding of the emergent behavior of complex biological systems [32,60,67]. Indeed, one of the points that can aid in the elucidation of the mechanisms assuring the functionality of any cellular process, is represented by the modeling and simulation of the spatio-temporal network of its molecular interactions.

By exploiting mathematical modeling tools and computer simulation techniques, integrated with the available experimental data, we are now in the condition of making predictions on a system behavior, therefore gaining insights into the working principles and the organization of various biological systems. In particular, computational tools allow to conduct many experiments that are unfeasible in laboratory. In this context, simulators specifically built for biological systems, which can be used to understand how the modeled system behaves in normal conditions and how it reacts to (simulated) alterations of some of its components, surely present many advantages over conventional experimental biology in terms of cost, ease to use and speed.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 119–143, 2010.

In the last few years a wide variety of models of cellular processes have been proposed, based on different formalisms. For example, deterministic models of chemical kinetics – based on the law of mass action – attempt to represent cellular processes by means of sets of ordinary differential equations (ODEs). At a different level of abstraction, many formalisms originally developed by computer scientists to model systems of interacting components, have been applied in the field of Computational Biology (e.g. Petri Nets [16] or π-calculus [55]).

Besides this, several experimental investigations have recently evidenced the presence of biological noise at the single-cell level [8,20,39], which is due to the inherently random interactions between cellular molecules and takes place especially for those species occurring in low amounts inside the cell. The classical modeling approach based on ODEs is not actually able to capture the effects of these stochastic processes, which can lead the cell behavior to complex dynamics such as bistability (that is, the possibility of switching between two different stable steady states), that can be effectively investigated only by means of stochastic modeling approaches. As a matter of fact, many algorithms developed to perform stochastic simulations of biological systems have proved their suitability for reproducing the dynamics of many cellular processes (see, e.g., [12,40,61] and references therein).

Recently, also the framework of P systems [47,48] has been exploited in this field of research. The most attractive features of P systems, which make them a suitable framework for the modeling of biological systems, can be recognized in their peculiar characteristics: the compartmentalized space, defined by means of a distributed membrane structure, which allows to delimit distinct "reaction volumes" where different reactions can take place and operate onto local molecular species; the easy understandable formalism in the writing of reactions, which provides a good flexibility in the expression and a good comprehension of the model also to non specialists; the possibility to communicate chemicals among distinct volumes, which grants the flow of information from a local level to the global level of the system, and allows to mimic the movement of chemicals within cellular spaces; the application of reactions performed in parallel at the level of volumes, which gives the strong capability to keep track of the global functioning of the system, and so on.

In this paper, we describe the main features of *BioSimWare*, a software based on the modeling approach of P systems, which allows to analyze a broad variety of biological systems, ranging from cellular processes to population phenomena. BioSimWare implements several stochastic simulation algorithms for both single and multi-volume systems, as well as other useful tools (e.g. parameter estimation and parameter sweep application) for the automatic analysis of the system dynamics under different conditions.

The paper is structured as follows. In the next three sections we deal with the theoretical issues that stand at the basis of BioSimWare. In particular, in Section 2 we present the modeling approach, while in Section 3 we briefly describe the stochastic algorithms currently available in BioSimWare for the simulation of single and multi-volume systems. The analysis tools are discussed in Section 4, where, in particular, we show how the problem of parameter estimation for stochastic systems can be faced with the use of (population-based) optimization methods. In Section 5 we present some examples of

application of BioSimWare for the modeling and analysis of various biological dynamics, as bistability, stiffness, oscillations, and so on. Section 6 concludes the paper with some final remarks on our simulation environment.

2 Modeling Biological Systems with BioSimWare

In this section, we describe the modeling approach used in BioSimWare, by presenting the feature of compartmentalization and the description of reactions and molecular species. We refer to Section 5 for some examples of various biological models that have been studied with BioSimWare.

2.1 Compartmentalization

In BioSimWare the question of space is handled by providing the definition of a "compartmental geometry", whereby space is divided into different volumes, with no superimpositions between one another (note also that no specific geometrical properties, given by a system of coordinates, are defined in BioSimWare for the compartmentalized volumes). Each volume can correspond to a real cellular compartment (i.e. the cytoplasm, the nucleus, the mitochondria, etc.), or can represent a "virtual" compartment, a concept that can be exploited to model specific aspects of a biological system (e.g. virtual volumes might be used to describe the diffusion of molecular species [1]).

Adjacent (real or virtual) volumes can communicate with each other, thus mimicking the passage of chemicals through a cellular membrane or their spatial diffusion. Moreover, each volume in the compartmentalized space is assumed to satisfy the conditions – uniform molecular distribution and fixed size – stated in Section 3.1.

As BioSimWare has been used for the modeling not only of cellular processes, but also of ecological systems, and might as well be applied to any other generic system describable in terms of individual components interacting in a compartmentalized geometry, we highlight here that the concept of volume can be extended to represent different types of compartments. For instance, in the study of metapopulation systems – which are ecological models describing the interactions and the behavior of populations living in fragmented habitats – the compartments represent the patches (i.e. local areas) where individuals of a population live, grow and interact with other individuals (see [7,3,5]).

2.2 Species and Reactions

After the definition of the compartmentalized geometry, the biological system is modeled in BioSimWare by providing a set of molecular species and a set of chemical reactions inside each volume.

Molecular species can be either "elementary" chemicals, or "molecular complexes" formed by the interaction of other (elementary or complex) species.

The interaction between chemicals is described by using the formalism of biochemical reactions, having the general form $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_k S_k \rightarrow \beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_k S_k$, where $S_1, \ldots, S_k$ are distinct molecular species, and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ (with some α_i, β_i possibly equal to zero) represent the stoichiometric coefficients. In other terms, a reaction of this type represents the interaction between α_1 copies of

species $S_1, \ldots, \alpha_k$ copies of species S_k (called the *reagents*), which produces β_1 copies of species $S_1, \ldots, \beta_k$ copies of species S_k (called the *products*). Actually, in the modeling of real biological systems, reaction of second-order, at most, should be defined (that is, no more than two copies of the same or different molecular species can appear as reagents). The rational behind this is that the simultaneous collision and chemical interaction of more than three molecules at a time, has a probability to occur close to zero in real systems.

According to the definition of the stochastic algorithms used to simulate the biological systems modeled with BioSimWare (see Section 3), each reaction is also characterized by a numerical factor, i.e. a stochastic constant, that expresses the chemical and physical characteristics of that molecular interaction. Note that, in order not to violate the correctness of the simulation algorithms, the reactions cannot have any other form than the one given above. For instance, reactions that describe a signal transduction event through a cellular membrane, and which have as reagents some chemical belonging to an external compartment and some other chemical belonging to an internal compartment, should be transformed into an opportune sequence of legal reactions if one wants to execute the standard form of stochastic simulation algorithms.

Particular attention is to be taken when dealing with "communication reactions", which are the reactions that allow to move chemicals among the various volumes occurring in the model (e.g. diffusion events). In BioSimWare, this aspect is implemented by attaching a destination target to each communication reaction, specifying the volume to which its products should be sent, as it is done in the definition of classical P systems. The communicated chemicals will then be considered as belonging to the target volume only at the end of the simulation step, as better described in Section 3.2 (see also [4,12]).

Note that, when defining models of biological systems where the basic interacting components are not chemicals (e.g. metapopulations, populations of cell, etc.), the formalization of species and reactions does not require any modification in BioSimWare. What changes is simply the meaning that these variables assume: for instance, in models of metapopulations, they correspond to population species and the interaction between individuals.

3 Stochastic Simulations Algorithms for Single and Multi-volume Systems

Several stochastic algorithms are implemented in BioSimWare and can be chosen for the simulation of the dynamics of single and multi-volume systems. These includes SSA [22], tau leaping [10], adaptive tau leaping [11], DPP [49], τ-DPP [14] and Sτ-DPP [13]. In this section, we briefly review some of the methods available with BioSimWare for both single volume (Sections 3.1) and multi-volume systems (Section 3.2).

3.1 Single Volume Stochastic Simulation Algorithms

The *stochastic simulation algorithm* (SSA) allows to generate an exact reproduction of the temporal evolution of biochemical systems [21,22]. In order to guarantee its exactness – i.e. the logical equivalence to the Chemical Master Equation – some necessary

conditions for the application of SSA have to be satisfied: (i) the system must be contained within a single volume, whose physical conditions (pressure, temperature, etc.) are assumed to remain constant during the simulation – that is, the volume cannot shrink or increase, nor divide into other volumes, and no experimental variations are assumed to be done during the simulation; (ii) the molecular species are considered uniformly distributed inside the volume – that is, the volume is assumed to be well-mixed. The amount of each molecular species is represented by the copy number of molecules; (iii) molecular species can interact each other according to a given set of chemical reactions, each one specified by a stoichiometric relationship between reactants and products. Each reaction is also characterized by a stochastic constant (whose unit of measurement is t^{-1}) that reflects the physical and chemical properties of the species involved in that reaction, under the experimental conditions fixed for the system.

At each time step of the simulation, the state of the system is represented by a state vector whose elements correspond to the amount of each molecular species occurring in the system. The state vector, together with the stochastic constants of reactions, is used to compute the probability for each reaction (its *propensity function*) to occur in the system at that time step. When a reaction event occurs in the system, the state vector is updated by removing the reactants that have been consumed and adding the products that have been produced. So doing, the state vector changes from time to time, and the propensity functions of reactions vary accordingly. In [22] it is shown how to evaluate, during each iterative step of the algorithm, the current time increment τ and the index μ of the reaction that will be executed during τ. By iteratively finding τ and μ, SSA performs a *sequential* description of the behavior of the system, meaning that at each time increment only one reaction event occurs. This can result in a huge computational burden, making therefore hard to efficiently simulate real cases of large biochemical systems.

In order to speed up stochastic simulations, in [23] it was introduced another SSA-based algorithm, called *(explicit) tau leaping*. With this method, instead of describing the dynamics of the system by tracing every single reaction event, a time increment τ is computed and a certain number of reactions are selected and executed in parallel. In this case, the obtained behavior of the chemical system is not exact, though the approximation error can be controlled by opportunely setting some parameters (see [10] for more details).

Several different versions of the tau leaping algorithm have been proposed, aimed at improving the procedure to compute the τ value and to select the reactions to be applied during each step. In BioSimWare, we have implemented the version of tau leaping defined in [10], where a good approximation of the system dynamics as well as the computational speedup are guaranteed by: (i) the "leap condition", used to uniformly bound the changes in the propensity functions of the reactions during each step; (ii) the division of reactions into "critical" and "non-critical" reactions, which avoid the possibility of generating state vectors with negative molecular populations (that might be due to the application of a number of reactions greater than the number of the necessary reagent molecules currently present inside the volume). A full description of the functioning and settings of tau leaping algorithm can be found in [10].

Some variants of tau leaping have also be presented to account for the phenomenon of stiffness in biochemical systems [24,25]. Stiff systems are characterized by the presence of widely varying timescales, that is, reactions with fast and slow timescales – with the fastest dynamical modes being stable. Indeed, many cellular systems consist of a mixture of fast reactions, which occur very frequently, and of slow reactions, which occur only rarely. The problem of stiffness is well known for ODEs, but it also impacts the performance of stochastic simulation algorithms. When simulating stiff systems, the SSA will spend most of the time in executing the fast reactions, one at a time; though, as the majority of the fast reactions are usually not as important as the slow ones, then the simulation will proceed very slowly (see the example given in Section 5.3). The same problem arises with tau leaping, since the selection procedure of the time steps – that assures the accurateness of the algorithm – is restricted to the timescale of the fastest reactions, thus generating unnecessarily small step sizes.

To overcome these limits, other variants of stochastic simulation algorithms have been developed for simulating stiff chemical systems: the *implicit tau leaping* algorithm [54] (which mirrors the implicit Euler method in ODE theory), the *slow-scale SSA* [9] (in which the fast reactions are skipped over), and the *adaptive explicit-implicit tau leaping* algorithm [11]. BioSimWare implements the adaptive algorithm, that is based on an opportune strategy to identify stiffness and to automatically switch between the explicit and the implicit selection methods of time steps in tau leaping, therefore achieving much better efficiency than other simulation methods.

3.2 Multi-volume Stochastic Simulation Algorithms

As stated above, the definition of standard stochastic simulation algorithms only holds for biochemical systems enclosed in a single volume. In order to overcome this limitation, novel approaches have been recently introduced. For instance, to simulate systems with spatial heterogeneity (i.e. non uniform distribution of chemicals), the next subvolume method [19] or the binomial τ-leap spatial stochastic simulation algorithm [37] can be used (see also [12] for a more detailed state of the art on this topic). These methods work by dividing the system volume into separated subvolumes, each one being small enough to be considered well-mixed. So doing, the requisites for SSA are satisfied, and both the evaluation of propensity functions and the diffusion processes occurring inside every single subvolume can be handled.

Another limitation consists in the fact that the volume occupied by the chemicals – or the fact that their amount can arbitrarily increase because of iterated application of some reactions – is not usually considered when performing stochastic simulations of biochemical systems.

To face these two questions – spatiality and the size of chemicals in stochastic simulation algorithms – in BioSimWare we have implemented two procedures, τ-DPP and Sτ-DPP, respectively, that are hereby presented.

τ-DPP. τ-DPP [14] is an algorithm based on tau leaping, that enables the simulation of multi-volume systems, where each volume is defined such that the usual SSA conditions hold. The distinct volumes can be arranged according to a specified hierarchy, under the additional assumption that the topological structure and the volume dimensions do not

change during the system evolution. Inside each volume, two different types of reaction can be defined: (1) *internal reactions* correspond to the classical biochemical reactions that modify the chemicals inside the volume where they take place; (2) *communication reactions* are used to send the chemicals from the volume where they are executed towards an adjacent volume. Both types of reactions have the form given in Section 2.2, and are characterized by a stochastic constant.

With τ-DPP, the temporal evolution of the whole system is generated by letting all volumes evolve in parallel, and by using the following strategy for the choice of time steps. At each iteration, we consider the state vector of each volume and calculate a time increment inside each volume independently from the others, according to the standard tau leaping algorithm. Then, the smallest time increment among all volumes is selected and used to evaluate the next-step evolution of the entire system. Since all volumes locally evolve according to the same time increment, τ-DPP is able to correctly work out the global dynamics of the multi-volume system, without causing any troubles in the time increment procedure within distinct volumes. In other words, we let all volumes proceed along a shared timeline, therefore avoiding paradoxical situations where one volume will execute reactions that take place in the future or in the past time of another volume. Moreover, by adopting this procedure, the simulated evolutions of all volumes get synchronized at the end of each iterative step. This also guarantees that the communication of chemicals among adjacent volumes is correctly simulated. A more detailed description of τ-DPP algorithm can be found in [14,4].

Sτ-DPP. Sτ-DPP [13] is a variant of τ-DPP that allows to characterize biochemical systems by adding appropriate measures to represent the size of the volumes and of the different molecular species within each volume. These measures are useful to avoid the possibility of having an infinite accumulation of chemicals inside a compartment, which can represent an unwanted feature in the modeling of some real biochemical system. In Sτ-DPP, the topological organization of the volumes is described by means of a hybrid membrane structure, where tree-like membrane structures [47] can be embedded into tissue-like membrane structures [38]. The communication of chemicals among volumes does not necessarily pertains only to adjacent volumes – as in classical membrane structures – but can be arbitrarily defined. This characteristic of Sτ-DPP takes inspiration from specific components of living cells, the so-called *microtubules* [52], to the aim of reproducing their role as intracellular "highways" for the transport of some cellular components, such as vesicles and proteins. The simulation of the dynamics with Sτ-DPP is then performed by applying the same strategy defined for τ-DPP, with an additional requirement on the application of reactions: the execution of a tossed reaction is allowed only if the production of chemicals does not exceed the available amount of "free volume" occurring in the current state (see [13] for more details and examples of application).

4 Tools for the Analysis of Stochastic Simulations

Computational investigations of biological systems require the knowledge of many numerical factors, like the molecular amounts and stochastic constants (concentrations of

molecular species and values of reaction rates, respectively, in a continuous deterministic context), which represent an indispensable quantitative information to perform an analysis of the system behavior. Unfortunately, the experimental values of these factors are usually not available, since carrying out their measurements *in vivo* can be sometimes tangling or even impossible [56]. In a few cases, the values of some parameters can be assumed by fitting a mathematical derived dynamics against the concentration time series that result from *in vitro* measurements. In general, anyway, the lack and the inaccuracy of these information cause the problem of assigning the correct values to all parameters, in order to reproduce the expected dynamics in the best possible way.

In this section we discuss about the importance of automatic tools for the analysis of the dynamics of a biological system and we focus, in particular, on the problems related to the choice of parameters. In this context, we present two methodologies implemented in BioSimWare, called *parameter estimation* (PE) and *parameter sweep* (PSA), discussed in Section 4.1. Optimization methods can be used to tackle the calibration problem of PE, by minimizing a cost function which quantitatively defines how good is the system behavior obtained by using some predicted values of the parameters, with respect to the actual experimental dynamics (see [41,56] and references therein). In addition, once that the calibration of the parameters has been carried out, it might be interesting to study the effects that different combinations of parameters values can cause. This issue can be faced by means of PSA, an analysis method suitable to the exploration of very large and complex search spaces. Other analysis tools that are currently under development in BioSimWare are sketched in Section 4.2.

4.1 Parameter Estimation

The peculiarity of the PE approach implemented in BioSimWare is that it embeds the (forward) problem of performing stochastic simulations of the system dynamics, into the (inverse) problem of estimating its unknown parameters. More precisely, this method exploits the outcome of stochastic simulation algorithms to effectively evaluate the fitness function used by the optimization techniques that are at the basis of our PE tool. To better explain this matter, we refer to Figure 1, left side, where we show the comparison between an experimental dynamics (the smooth line called "target"), and a set of stochastic dynamics generated by changing the value of one parameter of the model, which results in pretty distinct outcomes. Besides the marked differences that various "estimated" dynamics can present with each other, the right side graphic shows that, when dealing with stochastic simulations it is also necessary to develop appropriate methods to correctly measure the "distance" between the experimental curve and the (best) estimated curve, in order to handle the problem of stochastic fluctuations.

In [6] we started our analysis of PE by comparing the performances of two different optimization algorithms, genetic algorithms (GAs) [28] and particle swarm optimization (PSO) [31]. That work was motivated by the fact that PE represents an example of dynamic optimization problems, meaning that the fitness function value may change during the iteration of the algorithm – more precisely, each individual can have slightly different fitness values each time it is evaluated – and a number of contributions exist about the use of GAs and PSO for this kind of problems. In [6], both GAs and PSO

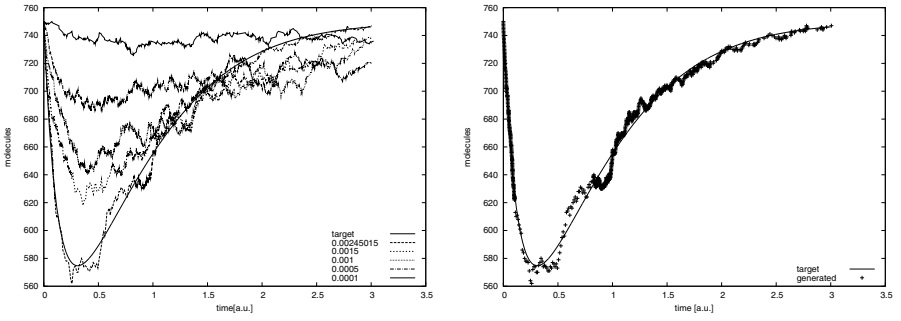


Fig. 1. *Left side*. Comparison between a target dynamics and some generated dynamics using different values of one parameter. *Right side*. Comparison between the best estimated stochastic dynamics and the target dynamics: the problem of stochastic fluctuations.

formulations used for PE evolve individuals consisting of n-length vectors of floating point numbers, where n is the number of parameters (e.g. stochastic constants) that have to be optimized.

The definition of the fitness function used in BioSimWare for running GAs and PSO, is based on the consideration that, in order to optimize the stochastic constants of a model (or, similarly, the amounts of molecular species), it is usual to compare a given experimental outcome (the target dynamics) with a set of estimated dynamics, that are generated by running a stochastic simulation algorithm using the parameter values codified in the individuals of GAs or PSO. In our implementation, the fitness of each individual is evaluated by calculating the *area* between the target dynamics and the estimated dynamics of each molecular species which has a known behavior, at the same time managing all the troublesome aspects that are inherent to stochastic simulations (such as different time samplings among the target and the estimated curves, the quantitative differences among stochastic simulations even if they are performed starting with the same initial conditions, etc. – see details and a complete explanation in [6]).

In BioSimWare different versions of these optimization techniques have been already implemented: we refer to [6,12] for further details and to Section 5.2 for a simple application of PE with this method.

Parameter sweep application. The methodology used in PE to determine the "goodness" of a set of parameters can also be exploited for PSA, a suitable method for the exploration of the parameter space of biological systems. Real systems usually consists of a large number of chemical species, which interact through many reactions, therefore resulting in a search space – constituted by all possible combinations of the parameters values – that has a high number of dimensions. Moreover, the behavior of these dynamical systems is mainly influenced by the fixed initial conditions, i.e. the molecular quantities and the value of stochastic constants – whereby different inputs can generate different dynamics outcome. So, if one wishes to quantify the influence of the parameter values on the system dynamics, the exploration of such kind of search spaces turns out to be a hard task, as it requires a huge number of simulations. PSA represents

one of the simplest techniques for facing this problem and, in addition, it can easily exploit high–throughput computing applications on parallel and distributed architectures.

To be more precise, PSA works by comparing the dynamics resulting from a chosen input setting with respect to a target dynamics, obtained by using a given parametrization. Hence, given the biological model to analyze, the first step consists in defining the ranges and the distribution of the parameters values that will be perturbed during the PSA, therefore finding an efficient method to sample the multidimensional space. In [42] this has been done by exploiting the quasi-random series, that allow to uniformly cover the search space with few samples. The quality of each initial parametrization of the model is then measured by using an appropriate fitness function (as the one described above). Since each instance of a PSA – corresponding to a simulation of the biochemical system – is independent from the others, the grid computing framework constitutes a good solution to tackle the high computational cost of this kind of application.

4.2 Other Analysis Tools

Other tools for the analysis of biological systems are currently under development in BioSimWare. The first tool concerns the automatic *analysis of steady states*: our approach exploits GAs to optimize a frequency distribution (e.g. gaussian distribution) of the values that a species undergoes during a simulation. So doing, we can determine the mean value of the steady state, the maximal and minimal values it assumes during stochastic fluctuations, its duration in time, etc. This kind of investigation is particularly interesting in case of systems showing bistability, whereby the dynamics can switch between two different steady states. In cases like this, we are interested in devising the measures characterizing the different steady states (mean value, amplitude), as well as the number of switches occurring during a fixed simulation time, the time spent in each state, etc. In Section 5.1 we briefly present an example of a prototype bistable scheme, the Schlögl system, in order to clarify the relevance of such issues.

The second tool concerns the application of *sensitivity analysis* (SA), which represents an indispensable technique to understand how much the variations of the model parameters impact on the variation of the model outcome (see [58, 57, 50, 27] and references therein). In other words, SA allows to measure the influence that the uncertainties occurring in the proposed model will have on the uncertainty in the results of simulations; as such, the development of SA tools will allow us to assess in a more accurate way the goodness of our models. Moreover, the analysis of sensitivities can also reveal which are the model factors bringing about the most striking effects on the system behavior: this is a very significant issue in the investigation of biological systems, since these factors can be assumed to be good "control points" of the system dynamics. Therefore, SA can also provide help in the design of ad hoc laboratory experiments.

The third tool that we are planning to develop concerns the *reverse engineering* of system networks, that is a relevant problem in the case that only scarce or partial information is available about the interactions of the various components of a system (e.g. which proteins interact with a given protein, what is the exact temporal cascade of protein-protein interactions, etc.). Our approach will consist in exploiting a

co-evolutionary system joining Genetic Programming [33] for the inference of the structure, and an optimization technique (as GAs or PSO) for the concurrent estimation of the system parameters.

5 Applications

BioSimWare has been exploited for the investigation of many biological systems, characterized by different levels of complexity in terms of the structure of the model and of its related dynamical behaviors. For instance, at the level of single cells we have investigated several cellular pathways, such as the Ras/cAMP/PKA pathway in yeast [15] and bacterial chemotaxis [2,1]. At the level of population of cells, we have proposed a study for the synthetic genetic oscillator called Repressilator, coupled with a quorum sensing communication mechanism in bacterial colonies [4]. At the level of ecosystems, we have analyzed several aspects of metapopulations [7,3,5]. In the following sections, we present a few examples of applications of BioSimWare, which highlight some of the relevant issues of modeling, simulation and analysis discussed above.

5.1 The Schlögl System

In this section we present one of the simplest prototype chemical system that presents a *bistable* dynamical behavior, the Schlögl system [63,66]. Bistability is a capacity exhibited by many biological systems, consisting in the possibility of switching between two different stable steady states in response to some chemical signaling (see, e.g., [17,51,65] and references therein). The Schlögl system consists of 4 chemical reactions and 3 molecular species: $r_1 : A + 2X \rightarrow 3X, r_2 : 3X \rightarrow A + 2X, r_3 : B \rightarrow 3X, r_4 : X \rightarrow B$, where A, B are chemical species given as input and always kept at a constant amount, while X is the species that exhibits the bistable behavior. The values of stochastic parameters used for the simulations presented in this section are: $A = 1 \cdot 10^5, B = 2 \cdot 10^5, X = 250, c_1 = 3 \cdot 10^{-7}, c_2 = 1 \cdot 10^{-4}, c_3 = 1 \cdot 10^{-3}, c_4 = 3.5$.

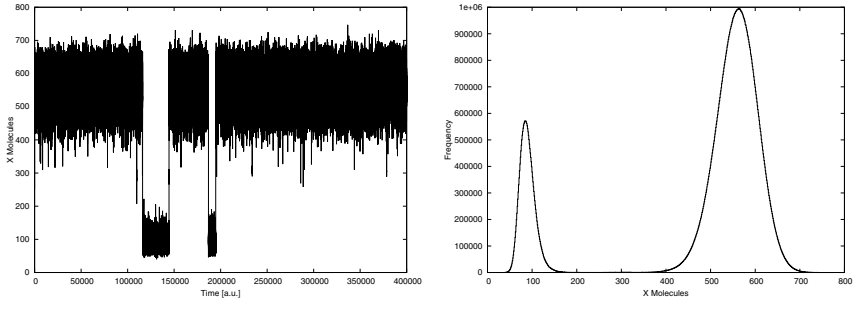


Fig. 2. *Left side*. Multiple switches between the upper and lower steady states in the Schlögl system. *Right side*. Frequency distribution of molecules X related to the left side simulation.

In Figure 2, left side, we show a simulation of the Schlögl system where four switches between the upper and lower stable states have occurred. On the right side, we plot the

frequency distribution of the values that species X assumes in the left side simulation, showing the bimodal behavior of the system (the peaks of distribution are centered on the medium values of the two steady states).

Bistable systems represent the most significant example in favour of stochastic modeling with respect to classical deterministic approach based on ODEs. Indeed, in cases like this the modeling with ODEs is not adequate, since it cannot capture the possibility of having random switches between the two stable steady state. As a matter of fact, in Figure 3 we compare one deterministic simulation (plain line) against four independent stochastic simulations of the Schlögl system (line and points). All dynamics were generated by considering the same initial conditions (the correspondence between stochastic and deterministic parameters has been determined by using the formula given in [22], and assuming a volume $V = 1.667 \cdot 10^{-17} l$). The graphic clearly shows that, starting from a given condition, the deterministic approach can only reach one of the two steady state, while stochastic simulations are able to represent both of the two possible behaviors of this bistable system.

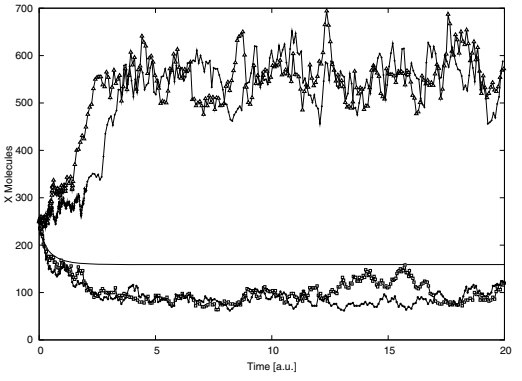


Fig. 3. Comparison between a deterministic model (ODE numerical integration, plain line) and a stochastic model (4 independent simulations, lines and points) of the Schlögl system

5.2 The Brussellator

The Brussellator is a simplified scheme for the Belousov-Zhabotinskii reaction, a family of inorganic redox reaction systems that exhibit macroscopic temporal oscillations and spatial patterns formation. This theoretical scheme is recognized as the prototype of nonlinear oscillating (open and well-stirred) systems, and well represents the significance and the variety of complex rhythms occurring in many biological systems. With respect to the formulation of the Brussellator given in [62], here we consider a description that leaves out the presence of two chemical products, since they are not directly involved in the formation of the oscillating limit cycle. Thus, we assume the following set of reactions: $r_1 : A \rightarrow X, r_2 : B + X \rightarrow Y, r_3 : 2X + Y \rightarrow 3X, r_4 : X \rightarrow \lambda$, where A, B are two chemicals that are given as input and always kept at a constant amount, X, Y are the intermediate product chemicals that exhibit oscillations, and λ represents the degradation of species X.

In Figure 4 we show how different values of the stochastic constants can lead to distinct dynamics for this simple system: the graphic on the left is characterized by regular oscillations, the one in the middle by initial oscillations that get rapidly damped (this behavior is obtained by a slight variation in the value of constant c_2, with respect to the previous simulation), the one on the right shows a random behavior, that is intermediate among the presence and the absence of oscillations (this behavior is obtained by varying the value of constant c_4, with respect to the first simulation). The exact values of constants $c_1, \ldots, c_4$ used to generate each dynamics are reported in the caption of Figure 4, while the initial amounts of molecules are $A = X = 200$, $B = 600$ and $Y = 300$ for all simulations.

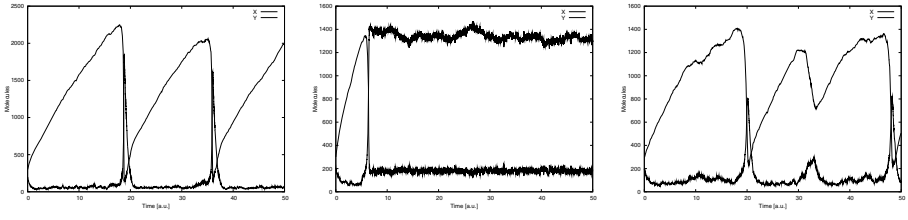


Fig. 4. Different dynamics of the Brussellator (thick line: species X, thin line: species Y). Values of stochastic constants: (*left*) $c_1 = 1, c_2 = 5 \cdot 10^{-3}, c_3 = 2.5 \cdot 10^{-5}, c_4 = 1.5$; (*middle*) $c_1 = 1, c_2 = 3.25 \cdot 10^{-3}, c_3 = 2.5 \cdot 10^{-5}, c_4 = 1.5$; (*right*) $c_1 = 1, c_2 = 5 \cdot 10^{-3}, c_3 = 2.5 \cdot 10^{-5}, c_4 = 1.5 \cdot 10^{-1}$.

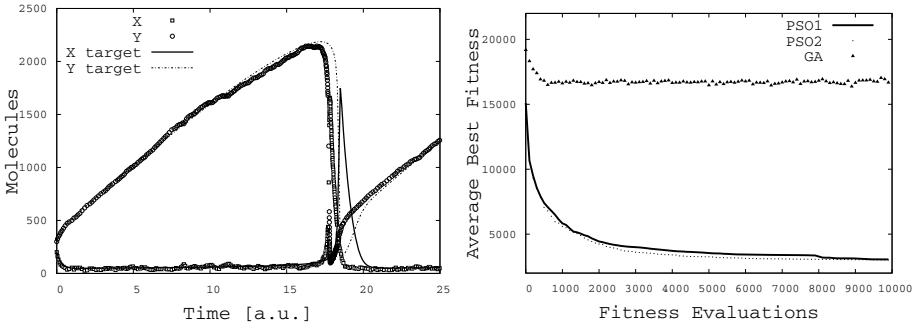


Fig. 5. Result of parameter estimation for the oscillating dynamics of the Brussellator. *Left side*. Comparison between target and estimated dynamics. *Right side*. Average Best Fitness against fitness evaluations.

In Figure 5 we show the results of the application of the PE method described in Section 4.1 to the Brussellator dynamics with oscillating regime, over one period of oscillation. On the left side, we compare the target dynamics – here generated by means of a transformation of the Brussellator in corresponding ODEs formulation – with an estimated dynamics, which has been generated considering the best solution found by PSO (the best solution corresponds to these estimated constants: $c_1 = 1.23966, c_2 = 0.00634, c_3 = 4.37171 \cdot 10^{-5}, c_4 = 2.71063$). On the right side, we report the Average

Best Fitness (ABF) against fitness evaluations for the implementation of GA and of the two variants of PSO that we have used for PE. We refer to [6] for further details and other examples of PE for simple biochemical systems.

5.3 Stiff Systems

In this section we show how the performance of stochastic simulation algorithms highly varies when dealing with the problem of stiffness. In particular, we compare SSA, tau leaping and adaptive tau leaping with respect to the average number of steps and to the execution time. To this aim, we consider a simple stiff system, the *decaying dimerization* (DD) model [23,11], which consists of 4 reactions and 3 molecular species: $r_1 : S_1 \rightarrow \lambda, r_2 : S_1 + S_1 \rightarrow S_2, r_3 : S_2 \rightarrow S_1 + S_1, r_4 : S_2 \rightarrow S_3$. The system models the irreversible isomerization (or decay) of species S_1, coupled with the reversible formation of complex S_2 (which mimics the dimerization of S_1), that can in turn be converted into the stable species S_3. If the values of constants c_2, c_3 are sufficiently large, then the decaying of S_1 through reaction r_1 is superimposed on the fast reversible dimerization given by reactions r_2, r_3. The values of stochastic constants and of initial molecular amounts for the simulations presented below have been chosen so that the reversible reactions r_2, r_3 are close to partial equilibrium (besides the initial steps), and they are: $c_1 = 1, c_2 = 10, c_3 = 1000, c_4 = 0.1, S_1 = 10000, S_2 = S_3 = 0$.

In Figure 6 we show the dynamics of the DD system obtained, under identical conditions, by means of SSA (top left), tau leaping (top right), adaptive tau leaping (bottom left), as well as the superimposition of these three dynamics (bottom right). As graphics show, all simulations have generated the same behavior for all molecular species. In particular, we can see that the initial population of S_1 immediately drops down because of reaction r_1, along with the rapid burst in the amount of S_2 due to the application of the concurrent reaction r_2. Thereafter, the unstable species S_1, S_2 start to be slowly depleted by the application of (slow) reactions r_1, r_4, respectively, while the population of S_3 slowly increases, thus turning the initial population of unstable S_1 molecules into a final population of stable S_3 molecules. The top graphics in Figure 6 also show the effect of slow and fast reactions during the execution of SSA and tau leaping algorithms: the simulation of the dynamics of species S_1 and S_2 requires a huge number of applications of the fast reactions r_2 and r_3 – which highlight the considerable stochastic fluctuations occurring in the dynamics of these species – but, as a consequence, turn out in a reduced performance of these two algorithms. If we are interested in efficiently capturing the dynamics of the most significant species in the DD model, i.e. S_3, without loosing computation time in simulating the exact dynamics of S_1 and S_2, then adaptive tau leaping should be used (bottom left), since it can more appropriately choose the length of simulation steps, therefore greatly enhancing its performance and, at the same time, resulting in dynamics that are very well comparable with the other algorithms (bottom right).

In Table 1 we compare the average number of steps and the execution times of the simulation of the DD model, evaluated over 10000 executions of SSA, tau leaping and adaptive tau leaping. All the simulations have been performed using a Personal Computer with an Intel Core2 duo CPU (2.66 GHz) running Linux (Ubuntu 9.10). The table clearly shows that, if compared with respect to the average number of executed steps,

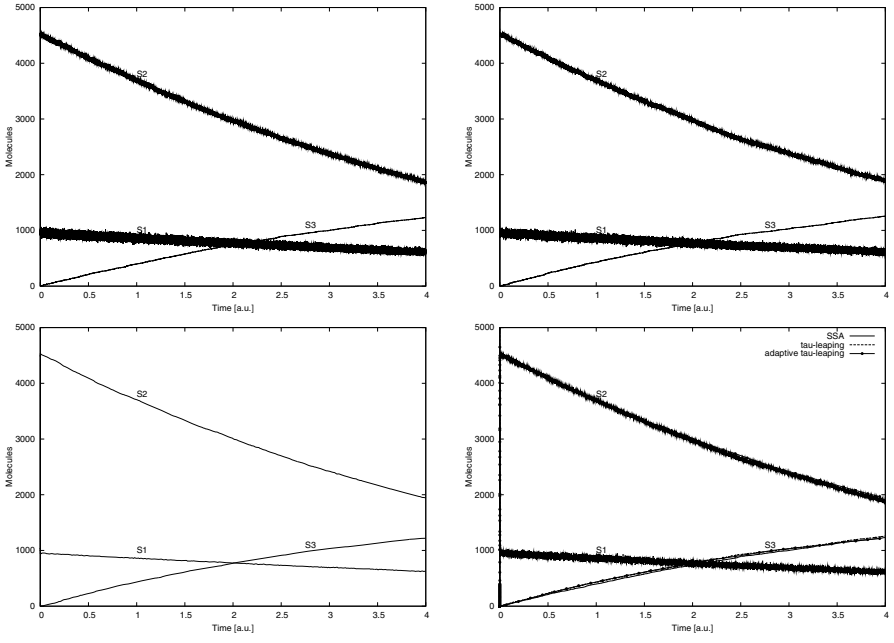


Fig. 6. Dynamics of the stiff decaying dimerization model (see text)

Table 1. Comparison of average number of steps and execution times between SSA, tau leaping and adaptive algorithm of the DD model

	SSA	**tau leaping**	**adaptive tau leaping**
average number of steps	$2.46 \cdot 10^7$	$1 \cdot 10^6$	945
total execution time	175h 1m	11h 47m	1m 51s

tau leaping is around 20-fold faster than SSA, and adaptive tau leaping is around 100-fold faster than tau leaping for this stiff system.

5.4 Bacterial Chemotaxis

Chemotaxis is an efficient signal transduction pathway, tightly regulated by feedback mechanisms, that allows bacterial cells to move in ever-changing environments in response to concentration gradients of attractants and repellents. The binding of ligand molecules to the chemotactic transmembrane receptors triggers a cascade of protein-protein interactions, which eventually influence the cytoplasmic amount of the phosphorylated form of a pivotal protein, CheY. CheYp rapidly diffuses through the cytoplasm and induces the rotations of flagella: if CheYp interacts with the proteins of the flagellar motor, then a clockwise (CW) rotation occurs, otherwise the flagellum will rotate counterclockwise (CCW). When the flagella are turning CW, they are uncoordinated and the bacterium performs a *tumbling* movement, while if *all* flagella are turning CCW, then

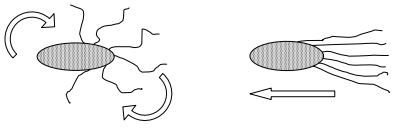


Fig. 7. Tumbling (left side) and running (right side) motions of bacterial cells after coordination of flagella

they get coordinated in a bundle, thus allowing the bacterium to swim directionally with a *running* movement (see Figure 7).

When they move in a homogeneous environment, bacteria usually perform random walks by alternating rapid tumblings with short runnings, which are determined by a high switch frequency of CW and CCW rotations of flagella. On the contrary, in the presence of a ligand concentration gradient, bacteria carry out longer directional swimming toward (against) the attractants (repellents), by reducing the switch frequency of flagella rotations. Anyway, if the ligand concentration remains constant in time, then the switch frequency is reset to the prestimulus level: the cell is able to realize an *adaptation* of its chemotactic response to the external variations of ligand concentration. So doing, bacteria can efficiently sample and adapt to the continuous mutations in their surroundings.

At the molecular level, this complex chemotactic response is governed by the (de)phosphorylation and (de)methylation of several transmembrane and cytoplasmic proteins, whose genetic regulation and biochemical functions have been well characterized [64]. Based on these data, in [2] we have defined a mechanistic model of the chemosensory system of *E. coli* bacteria, in response to attractant chemicals, consisting of 32 molecular species and 62 reactions, which accounts for all protein-protein interactions and the feedback control mechanisms regulating the pathway. Stochastic simulations have been performed with BioSimWare to analyze the temporal evolution of CheYp under different conditions, such as the deletion of other proteins involved in the pathway, the addition of distinct amounts of external ligand, and the effect of different methylation states (see [2] for a complete presentation of the model and simulation results).

In Figure 8, left side, we show a typical dynamics of CheYp in response to the simulated addition of two consecutive amounts of external ligand: starting from the steady state level of CheYp, the cell receives a first stimulus corresponding to a ligand amount of 100 molecules, added to the system at time $t = 3000$ sec and removed at time $t = 6000$ sec, and then a second stimulus corresponding to a ligand amount of 500 molecules, added at time $t = 9000$ sec and removed at time $t = 12000$ sec.

The results obtained through stochastic simulations have been exploited to investigate the possible effects that stochastic fluctuations of CheYp might have on the synchronization of flagella. In particular, we have focused on the analysis of the mean time periods during which the cell performs a running or a tumbling motion, as well as the time of adaptation to environmental changes, with respect to a varying number of flagella. This analysis has been carried out by using an automatic procedure that we defined to measure the time intervals of CW and CCW rotations of each flagellum. To distinguish between CW and CCW rotations, we assume that each flagellum is sensitive to a threshold level of CheYp, that is evaluated as the mean value (μ) of CheYp at

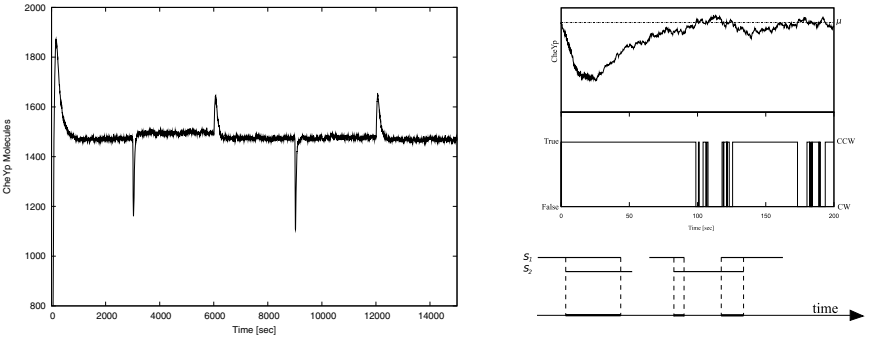


Fig. 8. *Left side*. Dynamics of CheYp: adaptation response to two consecutive stimuli. *Right side*. *(Top)* Determination of time intervals of CCW and CW rotation of the single flagellum ("true" and "false" values, respectively) with respect to the fixed threshold μ of CheYp. *(Bottom)* Synchronization of running motion between two different flagella (s_1, s_2 represent the time intervals of CCW rotation for flagellum 1 and 2, respectively).

steady state (see Figure 8, right side). Because of stochastic fluctuations, the amount of CheYp will randomly switch from below to above this value, thus reversing the rotation of each flagellum from CCW (value of CheYp below the threshold) to CW (value of CheYp above the threshold).

Then, to identify the periods of running motion, we determine the time intervals during which *all* flagella are rotating CCW. Similar considerations have been adopted for the analysis of tumbling motions and adaptation times. This analysis has been carried out for an increasing number n of bacterial flagella, with $n = 1, \ldots, 10$ (a typical *E. coli* cell possesses around half a dozen flagella). According to biological expectations, the analysis we performed in [2] on the effect of stochastic fluctuations have shown that: (i) the running-to-tumbling ratio decreases as n increases – this highlights the role of the number of flagella in the individual cell, and the necessity of their synchronization for a proper chemotactic behavior; (ii) the adaptation time does not depend strongly on the value of n in individual cells – this guarantees an appropriate adaptation mechanism, independently from phenotypic variations among distinct individual cells in a bacterial colony.

In [1] we extended this single volume model of bacterial chemotaxis to a multi-volume model, in order to account for the presence of a cytoplasmic gradient of CheYp, that is due to its diffusion from the area where it is phosphorylated (close to the chemotactic receptors) to the area of its activity (close to flagella) [36]. Its spatial localization, together with the localization of flagella, might indeed have a significant role in chemotaxis that is worth considering in this type of analysis. To this aim, we have split the bacterial volume into "slices", modeled by communicating virtual volumes, which allow to follow the diffusion of CheYp throughout the whole volume. So doing, in the analysis of synchronization of flagella, the virtual volumes containing a flagellar motor will be exposed to a local amount of CheYp that will be generally different from other local amounts in the adjacent virtual volumes. The analysis of flagella synchronization

in the multi-volume model will be presented in a forthcoming work, to the aim of determining whether significative changes to running, tumbling or adaptation times occur in the condition of CheYp diffusion.

5.5 Simulation of Fredkin Circuits by Chemical Reaction Systems

In this section we present one example of modeling and simulation of multi-volume systems, and refer to [1,5] for other multi-volume models analyzed with BioSimWare. Here we show how to simulate the functioning of logic gates and circuits, by exploiting formal chemical systems where molecular species and chemical reactions are used to perform computations. This is achieved by encoding boolean variables and logic functions into chemicals and reactions, respectively, and then following the temporal evolution of the chemicals to generate the correct mapping of the corresponding gate/circuit. The basic idea is to provide a "bottom-up" construction for the whole circuit, by using a single volume model to simulate each gate, and then opportunely connecting all the volumes such that the corresponding topology of the circuit is respected. So doing, the simulations of the multi-volume system compute exactly the logical function codified by the circuit itself. In particular, we show here the simulation of Fredkin gates and circuits.

The *Fredkin gate* is a boolean gate, whose map $\mathrm{FG} : \{0,1\}^3 \to \{0,1\}^3$ associates input triple $(\alpha_i, \beta_i, \gamma_i)$ to output triple $(\alpha_o, \beta_o, \gamma_o)$ according to the formula: $\alpha_o = \alpha_i$, $\beta_o = (\neg\alpha_i \wedge \beta_i) \vee (\alpha_i \wedge \gamma_i)$, $\gamma_o = (\alpha_i \wedge \beta_i) \vee (\neg\alpha_i \wedge \gamma_i)$. The Fredkin gate behaves as a conditional switch, where α_i can be considered a control line whose value determines whether the input values β_i and γ_i have to be exchanged or not: $\mathrm{FG}(1, \beta_i, \gamma_i) = (1, \gamma_i, \beta_i)$ and $\mathrm{FG}(0, \beta_i, \gamma_i) = (0, \beta_i, \gamma_i)$ for every $\beta_i, \gamma_i \in \{0,1\}$.

The simulation of the Fredkin gate can be done by defining two different types of molecular species for each of its lines, one species representing the boolean value 0, the other species the value 1. In particular, we use a, A for input α_i and output α_o on the first line (here two different chemical species suffice, since the output bit is always equal to the input bit on this line), b, B for input β_i and d, D for output β_o on the second line, c, C for input γ_i and e, E for output γ_o on the third line (since these two lines implement the logic switch, we need to distinguish among the 0 and 1 bits that are either given as input or generated as output). All these chemicals are manipulated inside a single volume by using 21 internal reactions: 6 *input reactions*, needed to produce the input chemicals inside the volume, thus simulating the input bits that are given to the gate; 10 *logical reactions*, which describe how the output chemicals are produced whenever the corresponding input chemicals appear inside the reaction volume; 5 *degradation reactions*, needed to avoid the simultaneous presence of two conflicting species inside the volume, representing both the states 0 and 1 on the same gate line. This problem may arise in the case that a second input is given to the logic gate after the first one: in this case, a time delay elapses before the corresponding output can be effectively generated – due to an average degradation time before the reaction volume can be cleared out of the *old* input and output chemicals – so that the *new* output chemicals can increase and produce the expected outcome. The complete model of the Fredkin gate is reported in [35].

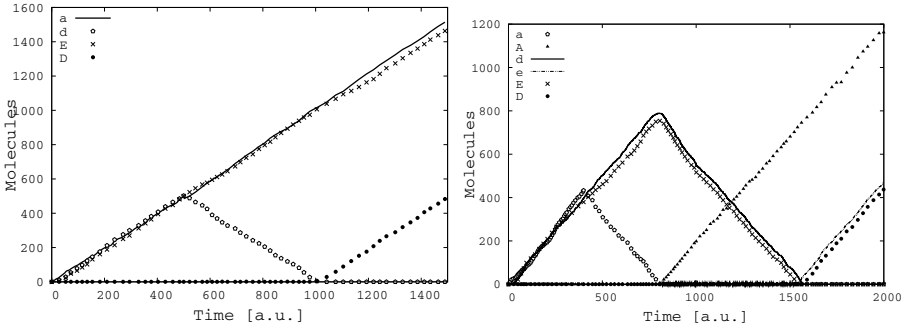


Fig. 9. Simulations of the Fredkin gate. *Left side*. Output results for the first input $(\alpha_i, \beta_i, \gamma_i) = (0, 0, 1)$ at $t = 0$, and the second input $(\alpha_i, \beta_i, \gamma_i) = (0, 1, 1)$ at $t = 500$. *Right side*. Output results for the first input $(\alpha_i, \beta_i, \gamma_i) = (0, 0, 1)$ at $t = 0$, and the second input $(\alpha_i, \beta_i, \gamma_i) = (1, 0, 1)$ at $t = 400$.

In Figure 9 we show two different simulations of the Fredkin gate, each one corresponding to two different input triples given at distinct time instants (see details in the caption). On the left side, we can see that, according to the functioning of the Fredkin gate, when the first input is received the corresponding output triple has to be $(\alpha_o, \beta_o, \gamma_o) = (0, 0, 1)$. Translated into molecular species, this output corresponds to the chemicals d and E produced inside the volume: this is simulated by the linear growth of these two species in the time interval $t \in [0, 500]$. At $t = 500$, when the second input is given, the molecular species D starts to be produced. Therefore, the system generates a configuration where some copies of both species d and D (values 0 and 1 on β_o) simultaneously occur inside the volume. To solve this contradiction, the output species d and D for β_o have to be degraded, along with the corresponding input species b and B for β_i. Once that all copies of species d are erased from the volume (around $t = 1000$), the number of copies of species D increases, thus generating the correct second output for the second line. Similar considerations can be done for the graphic shown on the right side of Figure 9, where an input that implements the logic switch between the second and third lines is simulated.

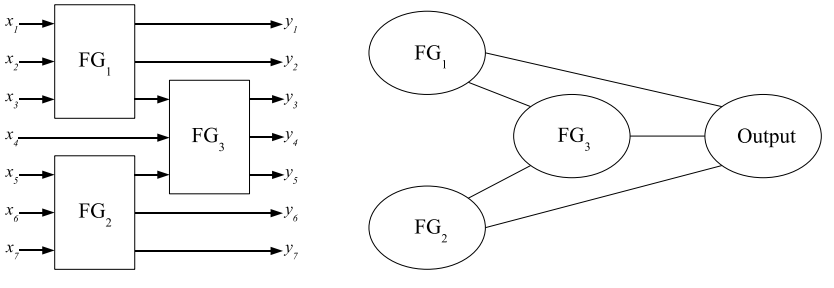


Fig. 10. A Fredkin circuit with three gates and two layers (*left*) and its tissue-like representation (*right*)

We consider now the circuit depicted on the left side of Figure 10, consisting of three Fredkin gates, structured into two layers (where we denote by $x_1, \ldots, x_7$ and $y_1, \ldots, y_7$ the input and output lines of the circuit, respectively). On the right side of this figure, we also represent the multi-volume model that is used here to simulate the entire circuit: we connect three reaction volumes (called FG_1, FG_2, FG_3) in such a way that the output chemicals of lines γ_o^1 and α_o^2 of gates 1 and 2 in the first circuit layer correspond to the input chemicals of lines α_i^3 and γ_i^3 of gate 3, respectively (where the superscripts on line names denote the number of the gate). Moreover, we consider an additional reaction volume whose function is to collect all output species of each gate.

Each reaction volume FG_i corresponds to a single volume model describing a Fredkin gate, with the addition of communication reactions inside volumes FG_1, FG_2, FG_3 in order to send the output chemicals coming from FG_1, FG_2, FG_3 to the output volume, and the output chemicals coming from FG_1 and FG_2 to volume FG_3. The output volume contains only the degradation reactions which are needed to erase the conflicting output chemicals whenever the input to the circuit is changed. The complete model of the Fredkin circuit is reported in [35].

In Figure 11 we show a simulation of this Fredkin circuit performed with BioSimWare by using the τ-DPP algorithm, where the output chemicals are split into three distinct panels for a better comprehension of their temporal evolution. The circuit receives

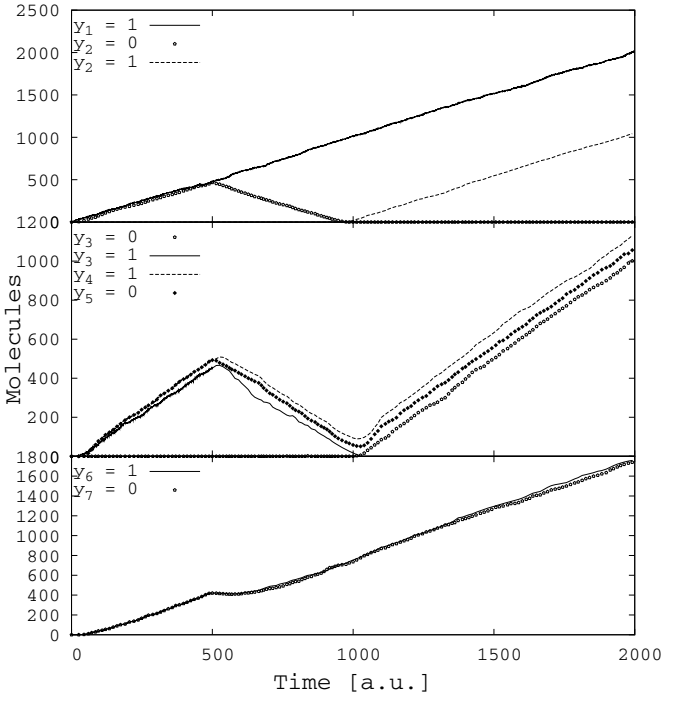


Fig. 11. Simulation of the Fredkin circuit: first input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 1, 0, 0, 1, 0, 1)$ given at $t = 0$; second input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 0, 1, 1, 0, 1, 0)$ given at $t = 500$

a first input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 1, 0, 0, 1, 0, 1)$ at time $t = 0$, and then a second input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 0, 1, 1, 0, 1, 0)$ at time $t = 500$. These two inputs generate the two outputs $(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (1, 0, 1, 1, 0, 1, 0)$ and $(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (1, 1, 0, 1, 0, 1, 0)$, respectively. Note that in the latter case only the bits on lines y_2 and y_3 are changed. During the first time interval $t \in [0, 500]$, the output chemicals corresponding to the first input are first generated inside the three reaction volumes FG_1, FG_2 and FG_3 and then, by using the communication reactions, all these output chemicals are collected in the output volume. At time $t = 500$ all lines of the circuit, except the first one, receive a different bit: this change results in the variation of two output bits on the lines y_2 and y_3, and the corresponding production of two new output chemicals in reaction volumes FG_1 and FG_3. The dynamics of these two output chemicals can be seen in the top and central panels, respectively, where it is apparent that the chemicals corresponding to the output bits $y_2 = 1$ and $y_3 = 0$ start to increase as soon as their conflicting chemicals (corresponding to the old output bits $y_2 = 0$ and $y_3 = 1$) are completely degraded.

6 Conclusion

The detailed modeling and stochastic simulation of the spatio-temporal cascades of molecular interactions is nowadays allowing a better comprehension of complex biological systems. Several disciplines, such as computer science, mathematics, physics, molecular biology, biotechnology, and so on, are therefore getting challenged in order to devise the theories or measurement/analysis tools that will provide the most appropriate combination of instruments to gain a deep understanding and a good control of biological systems. To this aim, several simulation environments have been and continue to be developed, based on stochastic or deterministic approaches, considering either spatial issues or well-mixed volume reactions, and providing different tools for the analysis of biological systems (see, e.g., [59,34]). In this context, BioSimWare represents one of the latest simulation environments, whose potentiality lies in the simplicity of use and in the numerous tools that it makes available for the stochastic investigation of various biological systems characterized by different levels of complexity, that can range from cellular processes to population phenomena or ecological systems.

BioSimWare is available for free download at http://biosimware.disco.unimib.it/. It can be run on single-processor personal computers (under Windows, Linux and Mac OS X operating systems), as well as on parallel architectures exploiting the message passing interface (MPI) [44], or on distributed architectures such as grid. Moreover, a further extension for graphic processing unit based on CUDA libraries is currently under development.

The software supports SBML format [29], and can also automatically convert stochastic models into the corresponding deterministic formulation. Simulations of systems of ODEs can be performed by using different numerical integration algorithms, provided by the gnu scientific libraries (gsl) [26], such as fourth order Runge-Kutta method.

A preliminary version of a user-friendly interface for BioSimWare has been developed by using Java language. Currently, the interface supports the modeling and

stochastic simulation of single volume systems by allowing: (1) the definition of the model through the data entry of reactions and parameters (stochastic constants, molecular amounts, species whose amount has to be fixed during the simulation, etc.); (2) the choice of the stochastic algorithm and of the simulation setting (total simulation time, value of the control parameter for the accuracy of tau leaping algorithm, etc.); (3) the creation and saving of the graphics corresponding to the simulated dynamics for the chosen molecular species; (4) the import and export of SBML files of the model.

Other P systems-based simulators, considering either the stochastic or the deterministic modeling approach, have been developed in the last years for the investigation of the dynamical properties of biological systems. We mention here the MetaPlab virtual laboratory [43], the Multicompartmental Gillespie framework [45], the workbench called Infobiotic [30], and the Cyto-Sim software [18]. We refer to [48] and [46] for further references, including several applications of P systems for the description of natural systems.

References

1. Besozzi, D., Cazzaniga, P., Cocolo, S., Mauri, G., Pescini, D.: Modeling diffusion in a signal transduction pathway: the use of virtual volumes in P systems. To appear in International Journal of Foundations of Computer Science
2. Besozzi, D., Cazzaniga, P., Dugo, M., Pescini, D., Mauri, G.: A study on the combined interplay between stochastic fluctuations and the number of flagella in bacterial chemotaxis. EPTCS 6, 47–62 (2009)
3. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Seasonal variance in P system models for metapopulations. Progress in Natural Science 17(4), 392–400 (2007)
4. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: A multivolume approach to stochastic modelling with membrane systems. In: Algorithmic Bioprocesses, pp. 519–542. Springer, Berlin (2009)
5. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: An analysis on the influence of network topologies on local and global dynamics of metapopulation systems. EPTCS 33, 1–17 (2010)
6. Besozzi, D., Cazzaniga, P., Mauri, G., Pescini, D., Vanneschi, L.: A comparison of genetic algorithms and particle swarm optimization for parameter estimation in stochastic biochemical systems. In: Pizzuti, C., Ritchie, M.D., Giacobini, M. (eds.) EvoBIO 2009. LNCS, vol. 5483, pp. 116–127. Springer, Heidelberg (2009)
7. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Modelling metapopulations with stochastic membrane systems. BioSystems 91(3), 499–514 (2008)
8. Blake, W.J., Kærn, M., Cantor, C.R., Collins, J.J.: Noise in eukaryotic gene expression. Nature 422, 633–637 (2003)
9. Cao, Y., Gillespie, D.T., Petzold, L.R.: The slow-scale stochastic simulation algorithm. Journal of Chemical Physics 122(1), 14116 (2005)
10. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. Journal of Chemical Physics 124, 44109 (2006)
11. Cao, Y., Gillespie, D.T., Petzold, L.R.: The adaptive explicit-implicit tau-leaping method with automatic tau selection. Journal of Chemical Physics 126, 224101 (2007)
12. Cazzaniga, P.: Stochastic algorithms for biochemical processes. Ph.D. thesis, Università degli Studi di Milano-Bicocca (2010)

13. Cazzaniga, P., Mauri, G., Milanesi, L., Mosca, E., Pescini, D.: A novel variant of tissue P systems for the modelling of biochemical systems. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 210–226. Springer, Heidelberg (2010)
14. Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G.: Tau leaping stochastic simulation method in P systems. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 298–313. Springer, Heidelberg (2006)
15. Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G., Colombo, S., Martegani, E.: Modeling and stochastic simulation of the Ras/cAMP/PKA pathway in the yeast Saccharomyces cerevisiae evidences a key regulatory function for intracellular guanine nucleotides pools. Journal of Biotechnology 133(3), 377–385 (2008)
16. Chaouiya, C.: Petri net modelling of biological networks. Briefings in Bioinformatics 8(4), 210–219 (2007)
17. Craciun, G., Tang, Y., Feinberg, M.: Understanding bistability in complex enzyme-driven reaction networks. Proceedings of the National Academy of Sciences 103(23), 8697–8702 (2006)
18. Cyto-Sim, `http://www.cosbi.eu/index.php/research/prototypes/cyto-sim`
19. Elf, J., Ehrenberg, M.: Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. IEE Proceedings Systems Biology 1(2), 230–236 (2004)
20. Elowitz, M.B., Levine, A.J., Siggia, E.D., Swain, P.S.: Stochastic gene expression in a single cell. Science 297, 1183–1186 (2002)
21. Gillespie, D.T.: General method for numerically simulating stochastic time evolution of coupled chemical-reactions. Journal of Computational Physics 22, 403–434 (1976)
22. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
23. Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. Journal of Chemical Physics 115(4), 1716–1733 (2001)
24. Gillespie, D.T.: Stochastic simulation of chemical kinetics. Annual Review of Physical Chemistry 58, 35–55 (2007)
25. Gillespie, D.T.: Simulation methods in systems biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 125–167. Springer, Heidelberg (2008)
26. The GSL Web Page, `http://www.gnu.org/software/gsl/`
27. Gunawan, R., Cao, Y., Petzold, L., Doyle, F.J.: Sensitivity analysis of discrete stochastic systems. Biophysical Journal 88, 2530–2540 (2005)
28. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975)
29. Hucka, M., et al.: The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4), 524–531 (2003)
30. The Infobiotic Web Page, `http://www.infobiotic.org/`
31. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of the IEEE International Conference on Neural Networks, Piscataway, NJ, vol. IV, pp. 1942–1948 (1995)
32. Klipp, E., Liebermeister, W., Wierling, C., Kowald, A., Lehrach, H., Herwig, R.: Systems Biology: A Textbook. Wiley, Chichester (2009)
33. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge (1992)
34. Lemerle, C., Di Ventura, B., Serrano, L.: Space as the final frontier in stochastic simulations of biological systems. FEBS Letters 579(8), 1789–1794 (2005)
35. Leporati, A., Besozzi, D., Cazzaniga, P., Pescini, D., Ferretti, C.: Computing with energy and chemical reactions. Natural Computing 9(2), 493–512 (2010)

36. Lipkow, K., Andrews, S.S., Bray, D.: Simulated diffusion of phosphorylated CheY through the cytoplasm of Escherichia coli. Journal of Bacteriology 187(1), 45–53 (2005)
37. Marquez-Lago, T.T., Burrage, K.: Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. Journal of Chemical Physics 127(10), 104101 (2007)
38. Martín-Vide, C., Pazos, J., Păun, G., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 573–679. Springer, Heidelberg (2002)
39. McAdams, H., Arkin, A.: Stochastic mechanisms in gene expression. Proceedings of the National Academy of Sciences 94(3), 814–819 (1997)
40. Meng, T.C., Somani, S., Dhar, P.: Modeling and simulation of biological systems with stochasticity. In Silico Biology 4, 24 (2004)
41. Moles, C.G., Mendes, P., Banga, J.R.: Parameter estimation in biochemical pathways: A comparison of global optimization methods. Genome Research 13(11), 2467–2474 (2003)
42. Mosca, E., Cazzaniga, P., Merelli, I., Pescini, D., Mauri, G., Milanesi, L.: Stochastic simulations on a grid framework for parameter sweep applications in biological models. In: Int. Workshop on High Performance Computational Systems Biology, HiBi 2009, vol. 0, pp. 33–42. IEEE Computer Society, Los Alamitos (2009)
43. The MP Virtual Laboratory, `http://mplab.scienze.univr.it/`
44. The MPI standard Web Page, `http://www-unix.mcs.anl.gov/mpi/`
45. P system modelling framework, `http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/~P_Systems_applications.htm`
46. The P Systems Web Page, `http://ppage.psystems.eu/`
47. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
48. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
49. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. International Journal of Foundations of Computer Science 17(1), 183–204 (2006)
50. Plyasunov, S., Arkin, A.: Efficient stochastic sensitivity analysis of discrete event systems. Journal of Computational Physics 221, 724–738 (2007)
51. Pomerening, J.R.: Uncovering mechanisms of bistability in biological systems. Current Opinion in Biotechnology 19(4), 381–388 (2008)
52. Pouton, C.W., Wagstaff, K.M., Roth, D.M., Moseley, G.W., Jans, D.A.: Targeted delivery to the nucleus. Advanced Drug Delivery Reviews 59(8), 698–717 (2007)
53. The PRISM Web Page, `http://www.prismmodelchecker.org/`
54. Rathinam, M., Petzold, L.R., Cao, Y., Gillespie, D.T.: Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. Journal of Chemical Physics 119, 12784–12794 (2003)
55. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Pacific Symposium of Biocomputing (PSB 2001), pp. 459–470 (2001)
56. Reinker, S., Altman, R.M., Timmer, J.: Parameter estimation in stochastic biochemical reactions. In: IEE Proceedings Systems Biology, vol. 153, pp. 168–178 (2006)
57. Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S.: Global Sensitivity Analysis: The Primer. Wiley Interscience, Hoboken (2008)
58. Saltelli, A., Ratto, M., Tarantola, S., Campolongo, F.: Sensitivity analysis for chemical models. Chemical Reviews 105, 2811–2827 (2005)
59. The SBML portal, `http://www.sbml.org/`
60. Szallasi, Z., Stelling, J., Periwal, V.: Systems Modeling in Cellular Biology. The MIT Press, Cambridge (2006)

61. Turner, T.E., Schnell, S., Burrage, K.: Stochastic approaches for modelling in vivo reactions. Computational Biology and Chemistry 28, 165–178 (2004)
62. Tyson, J.J.: Some further studies of nonlinear oscillations in chemical systems. Journal of Chemical Physics 58, 3919–3930 (1973)
63. Vellela, M., Qian, H.: Stochastic dynamics and non-equilibrium thermodynamics of a bistable chemical system: the Schlögl model revisited. Journal of the Royal Society Interface 6(39), 925–940 (2009)
64. Wadhams, G.H., Armitage, J.P.: Making sense of it all: bacterial chemotaxis. Nature Reviews Molecular Cell Biology 5(12), 1024–1037 (2004)
65. Widder, S., Macía, J., Solé, R.: Monomeric bistability and the role of autoloops in gene regulation. PloS One 4(4), e5399 (2009)
66. Wilhelm, T.: The smallest chemical reaction system with bistability. BMC Systems Biology 3(1), 90 (2009)
67. Wilkinson, D.J.: Stochastic Modelling for Systems Biology. Chapman & Hall, Boca Raton (2006)

Modeling Population Growth of Pyrenean Chamois (Rupicapra p. pyrenaica) by Using P-Systems

Maria Angels Colomer[1], Santiago Lavín[2], Ignasi Marco[2], Antoni Margalida[3], Ignacio Pérez-Hurtado[4], Mario J. Pérez-Jiménez[4], Delfí Sanuy[5], Emmanuel Serrano[2], and Luis Valencia-Cabrera[4]

[1] Dpt. of Mathematics, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
colomer@matematica.udl.es
[2] Servei d'Ecopatologia de Fauna Salvatge (SEFaS), Departament de Medicina i Cirurgia Animals, Universitat Autònoma de Barcelona (UAB),
E-08193-Bellaterra (Barcelona), Spain
{Santiago.Lavin,Ignasi.Marco,Emmanuel.Serrano}@uab.cat
[3] Bearded Vulture Study & Protection Group
Apdo. 43 E-25520 El Pont de Suert (Lleida), Spain
margalida@inf.entorno.es
[4] Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n. 41012 Sevilla, Spain
{perezh,marper,lvalencia}@us.es
[5] Dpt. of Animal Production, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
dsanuy@prodan.udl.cat

Abstract. P systems provide a high level computational modeling framework which integrates the structural and dynamic aspects of ecosystems in a comprehensive and relevant way. In previous works,several ecosystems modeled by using P systems were presented. The good results obtained encourage us to study new ecosystems such as the one presented in this paper. Pyrenean Chamois (*Rupicapra p. pyrenaica*) is an ungulate species inhabiting the Catalan Pyrenees. In recent years, several diseases have caused a drastic decrease in the number of individuals. Since they provide significant economic contributions in the area and constitutes an important food resource for obligate and facultative scavengers, it is very interesting to provide a model in order to facilitate the management of their ecosystems.

1 Introduction

Modeling a biological system is usually very complicated because each biological process involves a large number of factors interacting with each other. Therefore the most common solution is to define it a scenario in which the number of variables and interactions between variables is very limited.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 144–159, 2010.

Most of the existing models for the study of population dynamics are based on differential equations (ODEs), but this approach has some drawbacks. When the number of species in a model is greater than two, the equations system proposed is so complex that it is usually solved using numerical methods. Besides, improvements on the performance of the models are generally obtained by the addition of ingredients, which in the case of ODEs means that the whole modeling process needs to start from scratch.

Computer models based on P system offer significant advantages: modularity, parallelism, and no limitation on the number of interrelated variables that evolve parallely. These properties make them very attractive for modeling complex ecosystems.

Each ecosystem has its own important peculiarities, thus trying to design an "universal" ecosystem model is not a good approach. The model should be adapted taking into account if a protected and endangered species is being studied, or if the ecosystem deals with an invasive species, or simply an endemic area.

Nevertheless, there are some aspects common to most ecosystems such as:

- They contain a large number of individuals and species.
- The life cycles of species inhabiting the ecosystem display several basic processes such as: feeding, growth, reproduction and death.
- These processes are repeated annually.
- The evolution often depends on the environment: weather, soil, vegetation, etc.
- The natural dynamics suffers modifications due to human activities.

These common features yield some requisites for the model from a computational point of view: many processes take place simultaneously, there is cooperation between individuals and elements of the ecosystem, partial synchronization among the dynamic evolution sub-ecosystems (for example, there could be adverse weather conditions some year, and this does not affect a single subecosystem, but has a global influence on the entire ecosystem), situations need to be restored annually.

These considerations lead to the definition of an appropriate modeling semantic context for the P system. In particular, a precise semantics of the multienvironment functional probabilistic P system with active membranes has been used to model two real ecosystems: One based on scavenger birds in the Catalan Pyrenees (Spain) [1] and another one based on the zebra mussel (*Dreissena polymorpha*) in the Ribarroja reservoir (Spain) [2]. In the first case, an endangered species is modeled and the purpose of the obtained model is the study of the evolution of the ecosystem under different scenarios to make the most appropriate management decisions for the conservation of species. The second study case corresponds to a completely different situation, the zebra mussel is an exotic species that has shown an excellent adaptation after its introduction in the reservoir. Its uncontrolled reproduction causes significant economic and ecological damage.

In both cases we have designed a simulator to validate the results so that managers have now two tools enabling them to perform virtual experiments under different conditions.

To show that the proposed modeling framework enables the study of a wide range of ecosystems, this paper presents a model for the study of the Pyrenean Chamois dynamics in the Catalan Pyrenees (Spain). It is an emblematic species that was in danger of extinction a few years ago and have an important role on the chain food of this area. *Aquila chrysaetos*

2 Pyrenean Chamois

It is a small ungulate living in the Pyrenees of great interest, not only from a hunting standpoint, but also naturalistic and touristic. At present the existing population in the Pyrenees is estimated to be 53,000 individuals. The status of the species has not always been so favorable, for example, in the late 60s the population decreased down to the edge of extinction due to indiscriminate hunting. Fortunately, National Hunting Reserves managed by the regional administration were created in order to save the species.

This species has no major predators in the Pyrenees, except for the brown bear (*Ursus arctos*), and the golden eagle (*Aquila chrysaetos*). However it is a species with a small growth rate compared with other species of ungulates. In recent years, the population has suffered from several epidemics of infectious keratoconjunctivitis and, more recently from a new disease associated to a Border Disease Virus (Genus Pestivirus, Familiy Flaviviridae) has affected some Pyrenean chamois populations. However, only the second one has been responsible for local population decreases greater than 80% (Marco et al 2009). The Pyrenean chamois has a life expectancy of 20 years, though the mortality rate is high for animals older than 11 years. In early ages, the mortality rate ranges from 40% to 50%, while it is around 10% for adults younger than 11 years. At the age of two, they reach the reproductive age, and approximately 75% of females mate once a year generally producing one single descendant.

The disease associated to a Pestivirus is having a very important impact on a social and economic scale in the Pyrenees. The media have been blare of different epidemics occurring, being a reflection of the concerns of local communities, the Government of Catalonia, ranchers, hiking groups, conservationists and hunters. The suspension of Pyrenean chamois hunting in the affected areas has led to major loss of economic income. This loss is due not only to the lack of direct income through payment of hunting licences, but also by the disappearance of the indirect income (ecoturism) that hunters and their guests bring out. Last but not least, we must highlight the considerable ecological impact of the sudden disappearance of this herbivore in the affected areas. Despite the detailed studies being carried out currently, the resulting consequences in the ecosystem are still unclear.

In Spain, Pestivirus infection of the species is up to now affecting only the region of Catalonia. However, it is possible that the process spreads to other regions. The process also affects the French Pyrenees sector, which borders the affected Catalan area [6].

In the Catalan Pyrenees there are 5 main protected areas where Pyrenean chamois live in herds, figure 1. We assume that it is unlikely that this species moves between areas and hence, wherever there is a lack of resources the animals die.

In this work we aim to present a model to simulate the evolution of the Pyrenean Chamois populations in the Catalan Pyrenees.

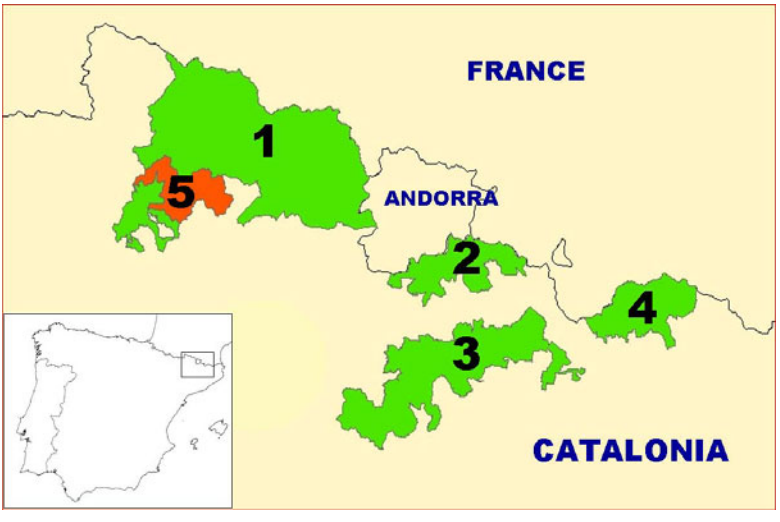


Fig. 1. Study area in the Catalan Pyrenees. Area 1: National Reservoir of hunting in l'Alt Pallars-Aran. Area 2: RNC Cerdanya-Alt Urgell. Area 3: RNC Cadí. Area 4: RNC Freser-Setcases. Area 5: Parc Nacional, not include in the study.

3 A P System Based Modeling Framework

It will define the variant of P-Sytem to be used for modeling the Pyrenean Chamois.

Definition 1. *A multienvironment functional probabilistic P system with active membranes of degree* (q, m) *with* $q \geq 1$, $m \geq 1$, *taking* T *time units,* $T \geq 1$, *is a tuple*

$$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,j} : r \in R_\Pi, 1 \leq j \leq m\}, \{\mathcal{M}_{i,j} : 0 \leq i \leq q-1, 1 \leq j \leq m\})$$

where:

- $G = (V, S)$ *is a directed graph such that* $(x, x) \in S$, *for each* $x \in V$. *Let* $V = \{e_1, \ldots, e_m\}$ *whose elements are called environments;*
- Γ *is the working alphabet and* $\Sigma \subsetneq \Gamma$ *is an alphabet representing the objects that can be present in the environments;*

– R_E *is a finite set of communication rules between environments of the form*

$$(x)_{e_j} \xrightarrow{p_{(x,j,j_1,\ldots,j_h)}} (y_1)_{e_{j_1}} \ldots (y_h)_{e_{j_h}}$$

where $x, y_1, \ldots, y_h \in \Sigma$, $(e_j, e_{j_l}) \in S$ $(l = 1, \ldots, h)$ *and* $p_{(x,j,j_1,\ldots,j_h)}(t) \in [0,1]$, *for each* $t = 1, \ldots, T$. *If* $p_{(x,j,j_1,\ldots,j_h)}(t) = 1$, *for each* t, *then we omit the probabilistic function. These rules verify the following:*

  ⋆ *for each* e_j *and for each* x, *the sum of functions associated with the rules from* R_E *whose left–hand side is* $(x)_{e_j}$ *coincide with the constant function equal to 1.*

– $\Pi = (\Gamma, \mu, R_\Pi)$ *where*

  • μ *is a membrane structure consisting of* q *membranes, with the membranes injectively labeled with* $0, \ldots, q-1$. *The skin membrane is labeled with 0. We also associate electrical charges from the set* $\{0, +, -\}$ *with membranes; and*

  • R_Π *is a finite set of evolution rules of the form* $r: \; u[\; v\;]_i^\alpha \to u'[\; v'\;]_i^{\alpha'}$ *where* $u, v, u', v' \in M(\Gamma)$, $i \in \{0, 1, \ldots, q-1\}$, *and* $\alpha, \alpha' \in \{0, +, -\}$;

– *For each* $r \in R_\Pi$ *and for each* j, $1 \le j \le m$, $f_{r,j}$ *is a computable function whose domain is* $\{1, 2, \ldots, T\}$ *and its range is contained in* $[0,1]$, *verifying the following:*

  ⋆ *For each* $u, v \in M(\Gamma)$, $i \in \{0, \ldots, q-1\}$ *and* $\alpha \in \{0, +, -\}$, *if* $r_1, \ldots, r_z$ *are the rules from* R_Π *whose left–hand side is* $u[v]_i^\alpha$, *then* $\sum_{j=1}^{z} f_{r,j}(t) = 1$, *for each* t, $1 \le t \le T$.

– *For each* j $(1 \le j \le m)$, $\mathcal{M}_{0,j}, \ldots, \mathcal{M}_{q-1,j}$ *are strings over* Γ, *describing the multisets of objects initially placed in the* q *regions of* μ.

A multienvironment probabilistic functional extended P system with active membranes of degree (q, m) taking T time units

$$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,j} : \; r \in R_\Pi, 1 \le j \le m\}, \{\mathcal{M}_{i,j} : \; 0 \le i \le q-1, \; 1 \le j \le m\})$$

can be viewed as a set of m environments $e_1, \ldots, e_m$ linked between them by the arcs from the directed graph G. Each environment e_j contains a functional probabilistic P system with active membranes of degree q, each of them with the *same* skeleton, Π, and such that $\mathcal{M}_{0j}, \ldots \mathcal{M}_{q-1,j}$ describe their initial multisets.

When a communication rule between environments

$$(x)_{e_j} \xrightarrow{p_{(x,j,j_1,\ldots,j_h)}} (y_1)_{e_{j_1}} \ldots (y_h)_{e_{j_h}}$$

is applied, object x pass from e_j to $e_{j_1}, \ldots, e_{j_h}$ possibly modified into objects $y_1, \ldots, y_h$, respectively. At any moment t, $1 \le t \le T$, in which an object x is in environment e_j, one and only one rule will be applied according to its probability which is given by $p_{(x,j,j_1,\ldots,j_h)}(t)$.

We assume that a global clock exists, marking the time for the whole system (for its compartments), that is, all membranes and the application of all rules are synchronized.

The tuple of multisets of objects present at any moment in the m environments and at each of the regions of the P systems located within them, and the

polarizations of the membranes in each P system, constitutes a *configuration* of the system at that moment. At the initial configuration of the system we assume that all environments are empty and all membranes have a neutral polarization.

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

4 Model

Pyrenean chamois is one of the species that were considered in the ecosystem modeled in [1]. However, that paper focuses mainly on the processes of feeding, reproduction, and mortality, including also that migration to other areas may occur in the event of resources shortage. The possibility that the species could be affected by disease was not taken into account, neither the fact that some biological parameters depend on the weather.

Taking advantage of the modularity of the P system, in order to cover these new aspects it suffices to modify the schema of the model presented in [1] by adding two modules, weather and disease module. In this work we have divided the mortality module in two, natural and hunter module. Besides, modules related to exchange between environments will be removed, since Pyrenean chamois do not migrate when there are not enough resources (as mentioned in section 2).

We present a preliminary study of the dynamics of Pyrenean Chamois, taking into account the following considerations:

- There are four separated areas in the Catalan Pyrenees were the Pyrenean chamois lives.
- Weather conditions, especially in winter (particularly the thickness of the snow layer), influence the values of biological parameters of the Pyrenean chamois species [4].
- Causes of death for this species include: natural death, hunting and disease. Only Pestivirus infection will be taken into account, while other diseases of importance that are known to affect the species will not be considered yet.
- The possibility of introducing more species in the model remains open. Note that in the same geographical space other wild and domestic ungulates may coexist in some cases, and this is worth studying especially if these species are competing for food with the parameters of the species.

The algorithmic scheme of the proposed model is shown in figure 2. The algorithm has been sequenced, but all animals evolve in parallel. The processes to be modeled will be the weather conditions (snow), reproduction, regulation of density, food, natural mortality, hunting mortality and mortality due to a disease. In order to model these processes for each species it is needed some biological, geographical and human factors, that is shown in Table 1.

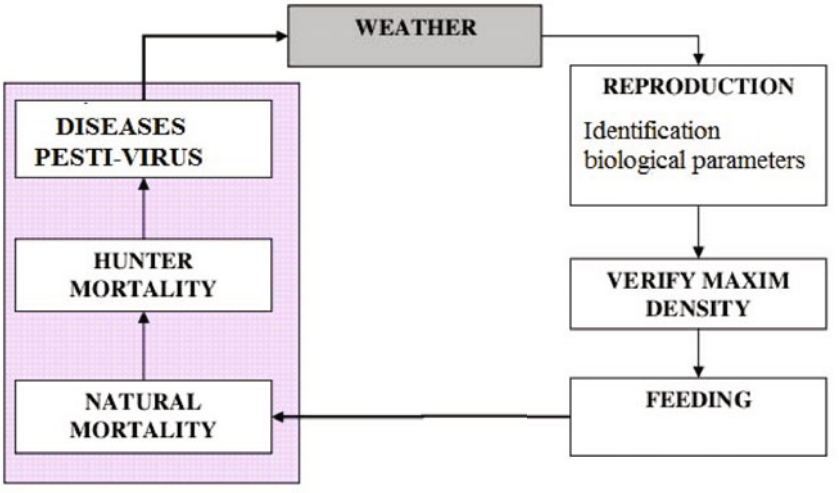


Fig. 2. Scheme model of the Pyrenean chamois model

The proposed model consists of a multienvironment functional probabilistic P system with active membranes of degree $(4, 11)$, taking T times units

$$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,\nu} : \ r \in R_\Pi, \ 1 \leq \nu \leq 4\}, \{M_{i,\nu} : \ 0 \leq i \leq 10, \ 1 \leq \nu \leq 4\})$$

where:

- The graph of the system is $G = (V, S)$, where $V = \{e_1, \ldots, e_4\}$ is the set of nodes called environments, and $S = \{(e_1, e_i) : \ 1 \leq i \leq 4\}$.
- The working alphabet is

$$\Gamma = \{X_{jy}, \ Y_{jy}, \ Y'_{jy}, \ Y''_{jy}, \ Z_{jy}, \ V_{jy}, \ W_{jy} : \ 0 \leq j \leq g_3, \ 1 \leq y \leq T\} \cup \{a, c, d, e, t, h, d_1 \ F, D, S, N\} \ \cup \ \{t_i : \ 1 \leq i \leq 3\} \ \cup \{G_i : \ 4 \leq i \leq 10\} \cup \{R_i : \ 0 \leq i \leq 7\}$$

  The objects X, Y, Y',Y'', Z, V and W are associated with animals in different states, index j represents the age of the animal and index y represents the moment of the simulation. The t are objects associated with the weather. F is an object that allows the generation of food in the form of grass. G_i are objects associated with the production of grass in the month i. The objects D, c, and e are used to control the density of animals of each species. The objects h_1 and h are used in order to know the state of Pesti-virus. The objects S and N indicate presence or absence of the disease, respectively, and finally there is the counter R that will allow us to synchronize the P System.
- The environment alphabet is

$$\Sigma = \{t, t_i : \ 1 \leq i \leq 10\}$$

Table 1. Biological and geographical information (i month, ν area, l Snow thickness category)

Biological	**Parameter**
Age at which they are considered adults	g_0
Age at which they begin to be fertile	g_1
Age at which they cease to be fertile	g_2
Life expectancy	g_3
Proportion of females in the population (as per 1)	k_1
Fertility rate (as per 1)	k_2
Number of descendants per female	k_3
Rate of natural mortality on young animals (as per 1)	m_1
Rate of natural mortality on adult animals (as per 1)	m_2
Amount of grass consumed per month and animal	$\beta_i\ 1 \leq i \leq 10$
Geographical	**Parameter**
Amount of grass produced per month	$\alpha_{i,\nu}, 1 \leq i \leq 10, 1 \leq \nu \leq 4$
Probability of having the disease	$ms_\nu, 1 \leq \nu \leq 4$
Probability of dying from a disease	$md_\nu, 1 \leq \nu \leq 4$
Maximum density of the ecosystem	$d1_\nu, 1 \leq \nu \leq 4$
Number of animals that survive after reaching the maximum density	$d2_\nu, 1 \leq \nu \leq 4$
Human factors	**Parameter**
Young animals hunted	$h1_\nu, 1 \leq \nu \leq 4$
Adult animals hunted	$h2_\nu, 1 \leq \nu \leq 4$

– The purpose of the set R_E of environment rules is to select the weather conditions for the year, and to distribute this information to all environments. This is done because there are some biological parameters that vary depending on weather conditions. In particular, we are able to simulate the snow thickness.

$$r_{e_1} \equiv (t)_{e_1} \xrightarrow{1/10} (t_i)_{e_1}(t_i)_{e_2}(t_i)_{e_3}(t_i)_{e_4}\ 1 \leq i \leq 10$$
$$r_{e_2} \equiv (t)_{e_k} \rightarrow (\#)_{e_k},\ 1 < k \leq 4$$

– $\Pi = (\Gamma, \mu, R_\Pi)$ is the skeleton of a functional probabilistic P system with active membranes of degree 11, whose membrane structure is $\mu = [\,[\,]_1[\,]_2 \ldots [\,]_{10}\,]_0$.

The set R_Π of rules of the system is the following (where the probabilistic constants associated with the rules have been incorporated):

* *Preparation of the system to start a cycle.*

$r_1 \equiv t_i[\,]_0^0 \rightarrow [t_i]_0^0,\ 1 \leq i \leq 10.$

$r_2 \equiv t_i[\,]_i^0 \rightarrow [t]_i^-,\ 1 \leq i \leq 10.$

After applying the environment rules, r_{e_1} object t_i enters from the environment carrying the information about the climatic condition of the next year to be simulated.

Each of the inner membranes labeled with 1, 2 or 10, stores information on biological parameters for each one of the then different climatic scenarios that the model envisages. The objects associated with animals should then enter the same membrane as object t_i.

$$r_3 \equiv X_{j,y}[\ ]_k^- \rightarrow [X_{j,y}]_k^0, \ \begin{cases} 1 \leq j \leq g_3, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

Each geographic area in which the species lives has a monthly production of food (grass).

$$r_4 \equiv \left(F[\ ]_k^- \rightarrow [G_4^{\alpha_4(\nu)}, \ldots, G_{10}^{\alpha_{10}(\nu)}]_k^0\right)_{e_\nu}, \ \begin{cases} 1 \leq k \leq 10, \\ 1 \leq \nu \leq 4. \end{cases}$$

where $\alpha_j(\nu)$ corresponds to the amount of grass produced in the area ν for the month j.

When the pesti-virus appears in an area, object h is produced. It will be present in the following configurations.

$$r_5 \equiv h[\ ]_k^- \rightarrow [h]_k^0, \ 1 \leq k \leq 10.$$

The amount of animals in the ecosystem should be controled so that it can not exceed a maximum load. This operation is performed by the objects a.

$$r_6 \equiv \left(c[\ ]_k^- \rightarrow [a^{0.9d1_\nu} e^{0.2d1_\nu}]_k^0\right)_{e_\nu}, \ \begin{cases} 1 \leq k \leq 10, \\ 1 \leq \nu \leq 4. \end{cases}$$

The following rules simulate the presence or absence of disease.

$$r_7 \equiv d[\ ]_k^- \rightarrow [d]_k^0, \ 1 \leq k \leq 10.$$

$$r_8 \equiv [d\ h \rightarrow d_1]_k^0, \ 1 \leq k \leq 10. \text{ ¡}$$

$$r_9 \equiv \left([d_1 \xrightarrow{ms_\nu} S]_k^0\right)_{e_\nu}, \ \begin{cases} 1 \leq k \leq 10, \\ 1 \leq \nu \leq 4. \end{cases}$$

$$r_{10} \equiv \left([d_1 \xrightarrow{1-ms_\nu} N]_k^0\right)_{e_\nu}, \ \begin{cases} 1 \leq k \leq 10, \\ 1 \leq \nu \leq 4. \end{cases}$$

Then we have counter R_i that will allow us to synchronize the P system

$$r_{11} \equiv R_0[\ ]_k^- \rightarrow [R_0]_k^0, \ 1 \leq k \leq 10.$$

$$r_{12} \equiv [R_i \rightarrow R_{i+1}]_k^0, \ \begin{cases} 0 \leq i \leq 4, \\ 1 \leq k \leq 10. \end{cases}$$

Finally, we introduce some randomness in the density control

$$r_{13} \equiv [e \xrightarrow{0.5} a]_k^0, \ 1 \leq k \leq 10.$$

$$r_{14} \equiv [e \xrightarrow{0.5} \#]_k^0, \ 1 \leq k \leq 10.$$

* *Reproduction rules*

Males of childbearing age

$$r_{15} \equiv [X_{j,y} \xrightarrow{1-k_1} Y_{j,y}D]^0_k, \begin{cases} g_1 \leq j < g_2, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

Females of childbearing age that reproduce

$$r_{16} \equiv [X_{j,y} \xrightarrow{k1 \cdot k2_l} Y_{j,y}Y^{k_3}_{0,y}D^{k_3+1}]^0_k, \begin{cases} g_1 \leq j < g_2, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

Females of childbearing age that do not reproduce

$$r_{17} \equiv [X_{j,y} \xrightarrow{k1 \cdot (1-k2_l)} Y_{j,y}D]^0_k, \begin{cases} g_1 \leq j < g_2, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

Animals that are not fertile

$$r_{18} \equiv [X_{j,y} \rightarrow Y_{j,y}D]^0_k, \begin{cases} g_2 \leq j \leq g_3, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

Young animals that do not reproduce

$$r_{19} \equiv [X_{j,y} \rightarrow Y_{j,y}D]^0_k, \begin{cases} 1 \leq j < g_2, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

* *Density rules*

Checking if the maximum density has been reached

$$r_{20} \equiv \left([D^{d1_\nu}a^{d1_\nu - d2_\nu}]^0_k \rightarrow [h_0]]^0_k\right)_{e_\nu}, \begin{cases} 1 \leq k \leq 10, \\ 1 \leq \nu \leq 4. \end{cases}$$

$$r_{21} \equiv [d, h_0]^0_k \rightarrow [d_0]^0_k, \ 1 \leq k \leq 10.$$

Transformation of objects that represent animals

$$r_{22} \equiv [Y_{j,y} \rightarrow Y'_{j,y}]^0_k, \begin{cases} 0 \leq j \leq g_3, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10 \end{cases}$$

* *Feeding rules*

$$r_{23} \equiv [Y'_{j,y}aG_4^{\beta_4}G_5^{\beta_5}G_6^{\beta_6}G_7^{\beta_7}G_8^{\beta_8}G_9^{\beta_9}G_{10}^{\beta_{10}} \rightarrow Z_{j,y}]^0_k, \begin{cases} 0 \leq j \leq g_3, \\ 1 \leq y \leq T, \\ 1 \leq k \leq 10. \end{cases}$$

where β_i represents the need of food in month i.

* *Natural mortality rules*

Young animals that survive

$$r_{24} \equiv \left([Z_{j,y} \xrightarrow{1-m1_{k,\nu}} V_{j,y}]_k^0\right)_{e_\nu}, \begin{cases} 0 \le j < g_0, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

Young animals that leave the ecosystem or die

$$r_{25} \equiv \left([Z_{j,y} \xrightarrow{m1_{k,\nu}} \#]_k^0\right)_{e_\nu}, \begin{cases} 0 \le j < g_0, \\ 1 \le y \le T, \\ 1 \le k \le 10 \\ 1 \le \nu \le 4. \end{cases}$$

Adult animals that survive

$$r_{26} \equiv [Z_{j,y} \xrightarrow{1-m2} V_{j,y}]_k^0, \begin{cases} g_0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10 \end{cases}$$

Adult animals that die

$$r_{27} \equiv [Z_{j,y} \xrightarrow{m2} \#]_k^0, \begin{cases} g_0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10. \end{cases}$$

Animals that reach the maximum age of the species

$$r_{28} \equiv [Y_{g_3,y} \to \#]_k^0, \begin{cases} 1 \le y \le T, \\ 1 \le k \le 10. \end{cases}$$

* *Hunting mortality*

Young animals that survive hunting

$$r_{29} \equiv \left([V_{j,y} \xrightarrow{1-h1_\nu} W_{j,y}]_k^0\right)_{e_\nu}, \begin{cases} 0 \le j < g_0, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

Young animals that are hunted

$$r_{30} \equiv \left([V_{j,y} \xrightarrow{h1_\nu} \#]_k^0\right)_{e_\nu}, \begin{cases} 0 \le j < g_0, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

Adult animals that survive hunting

$$r_{31} \equiv \left([V_{j,y} \xrightarrow{1-h2_\nu} W_{j,y}]_k^0\right)_{e_\nu}, \begin{cases} g_0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

Adult animals that are hunted

$$r_{32} \equiv \left([V_{j,y} \xrightarrow{h2_\nu} \#]_k^0\right)_{e_\nu}, \begin{cases} g_0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

* *Disease mortality*

$$r_{33} \equiv [R_5 S]_k^0 \to [R_6\ h]_k^-,\ 1 \le k \le 10.$$

$$r_{34} \equiv [R_5 N \to R_6\ h]_k^0,\ 1 \le k \le 10.$$

$$r_{35} \equiv [R_5 d_0 \to R_6\ h]_k^0,\ 1 \le k \le 10.$$

$$r_{36} \equiv [R_5 d \to R_6]_k^0,\ 1 \le k \le 10.$$

$$r_{37} \equiv [R_6]_k^- \to [\#]_k^+,\ 1 \le k \le 10.$$

$$r_{38} \equiv [R_6]_k^0 \to [\#]_k^+,\ 1 \le k \le 10.$$

$$r_{39} \equiv ([W_{j,y}]_k^- \xrightarrow{md_\nu} [\#]_k^+)_{e_\nu}, \begin{cases} 0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

$$r_{40} \equiv ([W_{j,y}]_k^- \xrightarrow{1-md_\nu} [W_{j,y}]_k^+)_{e_\nu}, \begin{cases} 0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10, \\ 1 \le \nu \le 4. \end{cases}$$

* *Updating rules*

$$r_{41} \equiv [W_{j,y}]_k^+ \to X_{j+1,y+1}[\ ]_k^0, \begin{cases} 0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10. \end{cases}$$

$$r_{42} \equiv [Y'_{j,y}]_k^+ \to [\#]_k^0, \begin{cases} 0 \le j < g_3, \\ 1 \le y \le T, \\ 1 \le k \le 10. \end{cases}$$

$$r_{43} \equiv [t]_k^+ \to R_0, F, t, c, d[\ ]_k^0,\ 1 \le k \le 10.$$

$$r_{44} \equiv [h]_k^+ \to h\ [\ ]_k^0,\ 1 \le k \le 10.$$

$$r_{45} \equiv [a]_k^+ \to [\#]_k^0,\ 1 \le k \le 10.$$

$$r_{46} \equiv [G_i]_l^+ \to [\#]_k^0, \begin{cases} 4 \le i \le 10, \\ 1 \le k \le 10. \end{cases}$$

$$r_{47} \equiv [t]_0^0 \to t[\ ]_0^0$$

– $\mathcal{M}_{0,\nu},\ \ldots\ \mathcal{M}_{10,\nu}$ $(1 \le \nu \le 4)$ are strings over Γ which describe the initial multiset of objects located in the regions of μ.

$$\mathcal{M}_{0,\nu} = \{F, R_0, c, d\} \cup \{X_{j,1}^{q_{\nu,j}} :\ 1 \le \nu \le 4,\ 1 \le j \le g_3\}, \quad \text{for } 1 \le \nu \le 4.$$
$$\mathcal{M}_{i,\nu} = \emptyset, \quad \text{for } 1 \le i \le 10 \text{ and } 1 \le \nu \le 4.$$

5 A Software Tool for Simulation

A software tool under GNU GPL license [8] for simulating P systems modeling ecosystems was presented in [2]. The simulation of two real and relevant ecosystems has been achieved by using this software tool [2]. One of them is related to an endangered species (the bearded vulture) and the other one is related to an exotic and invasive species (the zebra mussel). For each one, an *ad hoc* graphic user interface (GUI) has been developed in order to configure the initial parameters of the ecosystem and collect the results of the simulation by means of tables and graphics. The simulation core that has been used is based on P-Lingua [5,9] and pLinguaCore [5,10]. The software tool allows two different types of users: the first one is the *designer user*, who is the responsible for defining, debugging and validating the model for the ecosystem; and the second one is the *end-user*, who is the final user of the software tool and he/she uses it for carrying out virtual experiments over the ecosystem.

One of the main problems of the software tool is the need to design, develop and maintain several different graphic user interfaces. In this paper, a new software tool, *MeCoSim* [7], has been used. MeCoSim (Membrane Computing Simulator) allows the same functionality as its predecessor, and besides the *designer user* is provided with an easy-to-use method for creating new *ad hoc* GUIs for specific ecosystem models, by means of the definition in data bases. In this sense, the development of the GUI in Java Swing (or other programming languages) has been avoided, delegating this process on the designer user.

To summarize, MeCoSim offers to the users (designer and end-user) a highly customizable simulators generator to apply simulation algorithms for P systems modelling several scenarios under study. Thus, MeCoSim is a final product that avoid the necessity of ad-hoc GUI development per each scenario, introducing enough flexibility to permit the designer user to generate a simulator adapted to the scope of the domain of study of the end-user, with the inputs, parameters and outputs he needs.

The process to adapt MeCoSim to each scenario requires the definition of a configuration file. The structure of the file is provided to the designer user in order to configurate the custom simulator he wants to generate. After that, the file is processed by MeCoSim, that loads it in an embed database and generates the custom simulator that comply with the information introduced by the designer user. With this simple task done by the designer and without any software development, the end-user will get a custom simulator for his specific domain problem or case study. A change in the original model structure (desired structure of inputs, outputs or parameters) will be reflected in the simulator with the simple change of the configuration file and its reload in MeCoSim. For more information about the GUI configuration process with MeCoSim, see [7].

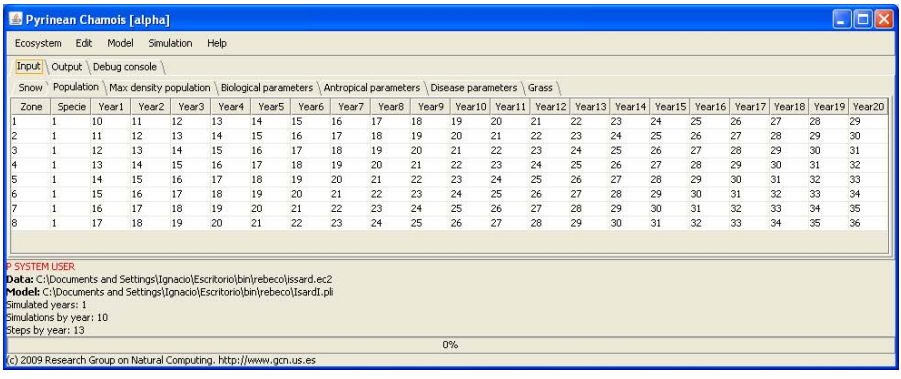


Fig. 3. MeCoSim3: edition process for the initial parameters of the model

Figure 3 shows a snapshot of the software tool MeCoSim with an specific graphic user interface for editing the initial parameters of the model presented in this paper.

6 Results

There are experimental data available from 1988 on although censuses where not carried out annually so that, experimental series is not a continuous one. Using the censuses in 1988 as input for the model, 22 years have been simulated repeating the process 50 times for each of the years simulated. Figure 4 shows the results and more specifically, the continuous line represents the average value of the 50 simulations whereas the broken lines correspond to 95% interval and the dots are the values obtained experimentally. In both Alt Pallars-Aran and Cerdanya-Alt Urgell areas, animals have suffered from pesti-virus infections whereas in Freser-Setcases, the population dynamics have not suffered the disease caused by this infection.

In general, the model behaves well in all cases; nonetheless, there are some experimental values near the lines corresponding to the confidence interval that should be studied.

The model considers the main processes and dynamics of the species although some of them have been omitted because they are considered to be less important. This may explain the differences between the values obtained with the model and experimental ones. Among these factors, it should be highlighted the influence of domestical animals living in the area on the spread of pesti-virus infection. In addition, there are few data regarding the thickness of the snow layer and those used in the model have been obtained from ski resorts so that they may be overdimensioned and then, may affect the results significantly. We suggest studying the relationship between the thickness of the snow layer and other available climatic data in the area such as temperature and the length of the winter interval.

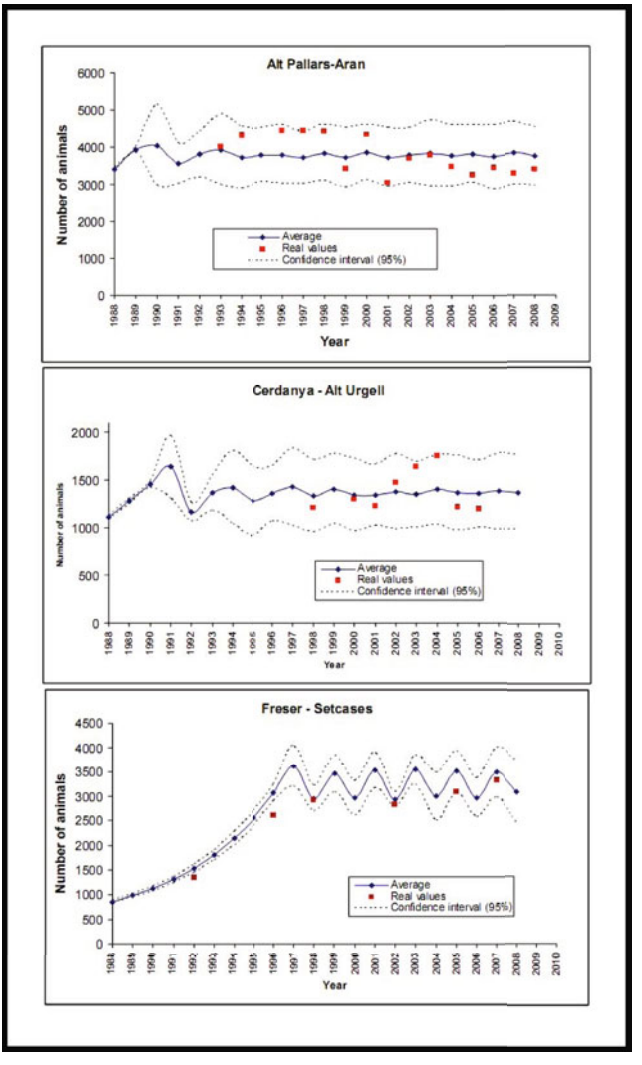


Fig. 4. Results

7 Conclusions

This paper presents the first computational model of a real ecosystem from the Catalan Pyrenees involving the Pyrenean Chamois. The model is based on Membrane Computing.

In [2], a general framework for modeling ecosystems using multienvironment P systems was presented. The rules in that framework were associated with probability functions depending on a number of parameters, such as the simulation time. The model presented in this paper can be considered as a practical

application of the introduced framework previously. The model is still under the experimental validation phase. Some preliminary validation of this model has been assessed by ecologists resulting in very encouraging and promising results that will be reported elsewhere. In order to assist the validation of the model, we are developing a simulation tool called MeCoSim [7]. This software allows the designer user to specify the Graphics User Interface for a specific model by editing a configuration file.

References

1. Cardona, M., Colomer, M.A., Pérez–Jiménez, M.J., Sanuy, D., Margalida, A.: Modelling ecosystems using P Systems: The Bearded Vulture, a case of study. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 137–156. Springer, Heidelberg (2009)
2. Cardona, M., Colomer, M.A., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D.: A Computational Modeling for real Ecosystems Based on P Systems. Natural Computing, http://dx.doi.org/10.1007/s11047-010-9191-3
3. Colomer, M.A., Margalida, A., Sanuy, D., Pérez-Jiménez, M.J.: A bio-inspired computing model as a new tool for modeling ecosystems: the avian scavengers as a case study. Ecological modelling (in press, 2010)
4. Crampe, J.P., Gaillard, J.M., Loison, A.: L'enneigement hivernal: un facteur de variation du recrutement chez l'isard (Rupicapra pyrenaica pyrenaica). Canadian Journal of Zoology 80, 306–1312 (2002)
5. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An overview of P-lingua 2.0. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 264–288. Springer, Heidelberg (2010)
6. Marco, I., Rosell, R., Cabezón, O., Mentaberre, G., Casas, E., Velarde, R., Lavín, S.: Border disease virus among chamois, Spain. Emerging Infectious Diseases 15, 448–451 (2009)
7. Valencia-Cabrera, L., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Colomer, M.A.: MecoSim: A General purpose software tool for simulating biological phenomena by means of P systems. In: Pre-proceedings of the IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010) (in press, 2010)
8. GPL license, http://www.gnu.org/copyleft/gpl.html
9. The P-Lingua website, http://www.p-lingua.org/
10. The pLinguaCore library website, http://www.p-lingua.org/wiki/index.php/PLinguaCore

On Generalized Communicating P Systems with One Symbol*

Erzsébet Csuhaj-Varjú[1,**], György Vaszil[1], and Sergey Verlan[2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende u. 13-17, 1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu
[2] Laboratoire d'Algorithmique, Complexité et Logique, Département Informatique
Université Paris Est
61, av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract. Generalized communicating P systems (GCPSs) are tissue-like membrane systems with only rules for moving pairs of objects. Despite their simplicity, they are able to generate any recursively enumerable set of numbers even having restricted variants of communication rules. We show that GCPSs still remain computationally complete if they are given with a singleton alphabet of objects and with only one of the restricted types of rules: parallel-shift, join, presence-move, or chain.

1 Introduction

The notion of a *generalized communicating P system* was introduced in [14], with the aim of providing a common generalization of various purely communicating models in the framework of P systems.

A generalized communicating P system, or a *GCPS* for short, corresponds to a hypergraph where each node is represented by a cell and each edge is represented by a rule. Every cell contains a multiset of objects which – by communication rules – may move between the cells. The form of a *communication rule* is $(a,i)(b,j) \to (a,k)(b,l)$ where a and b are objects and i,j,k,l are labels identifying the input and the output cells. Such a rule means that an object a from cell i and an object b from cell j move synchronously to cell k and cell l, respectively. Communication rules can also be interpreted as *interaction rules*.

Depending on their form, several *restrictions on communication rules* (modulo symmetry) can be introduced; we provide a detailed description of these variants in Section 2. When a GCPS has only one type of these restricted rules, then we

* Research supported in part by the Hungarian Scientific Research Fund, "OTKA", project K75952 and by Science and Technology Center in Ukraine, project 4032.

** Also affiliated with the Department of Algorithms and Their Applications, Faculty of Informatics, Eötvös Loránd University, Pázmány Péter sétány 1/c, 1117 Budapest, Hungary.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 160–174, 2010.

speak of a *generalized communicating P system with minimal interaction* or a *minimal interaction P system* (with the given type of rules), called also a GCPSMI, for short.

Due to the simplicity of their rules, the generative power of minimal interaction P systems is of particular interest and it has been studied in details. In [14,4] it was proved that eight of the possible nine restricted variants (with respect to the form of rules) are able to generate any recursively enumerable set of numbers; in the ninth case only finite sets of singletons can be obtained. Furthermore, these systems even with relatively small numbers of cells and simple underlying (hypergraph) architectures are able to achieve this generative power.

In this paper, we introduce one more restriction, i.e., we study minimal interaction P systems where the *alphabet of objects is a singleton*. We show that these so-called *one-symbol minimal interaction P systems* given with any of the *parallel-shift, presence-move, chain*, and *join rules* are able to generate every recursively enumerable set of natural numbers. These results demonstrate that computational completeness can be achieved with (certain variants of) GCPSMIs defined over the simplest alphabets. We also examine the generative power of one-symbol minimal interaction P systems with *conditional-uniport-in rules* and prove that these constructs are less powerful than the previous variants. We note that, by definition, the concept of the remaining restricted communication rules is not applicable for GCPSs over a one-symbol object alphabet. Finally, we provide topics for future research.

2 Preliminaries

In this section we recall some basic notions and notations used in membrane computing, formal language theory and computability theory. For further details and information the reader is referred to [9,10,11].

An alphabet is a finite non-empty set of symbols. For an alphabet V, we denote by V^* the set of all strings over V, including the empty string, λ.

A finite multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$; $M(a)$ is said to be the multiplicity of a in M ($\mathbb{N}$ denotes the set of non-negative integers). A finite multiset M over an alphabet V can be represented by all permutations of a string $x = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$, where $a_j \in V$, $1 \leq j \leq n$; x represents M in V^*. If no confusion arises, we also may use the customary set notation for denoting multisets. The size of a finite multiset M, represented by $x \in V^*$ is defined as $\Sigma_{a \in V} |x|_a$.

A *register machine* is a 5-tuple $M = (Q, R, q_0, q_f, P)$, where Q is a finite non-empty set, called the set of *states*, $R = \{A_1, \dots, A_k\}$, $k \geq 1$, is a set of *registers*, $q_0 \in Q$ is the *initial state*, and $q_f \in Q$ is the *final state*. P is a set of *instructions* of the following forms: $(p, A+, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called an *increment instruction*, or $(p, A-, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called a *decrement instruction*. Furthermore, for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q, s)$.

A *configuration* of a register machine M, defined above, is given by a $(k+1)$-tuple $(q, m_1, \dots, m_k)$, where $q \in Q$ and $m_1, \dots, m_k$ are non-negative integers, q

corresponds to the current state of M and $m_1, \ldots, m_k$ are the current numbers stored in the registers (in other words, the current contents of the registers or the value of the registers) $A_1, \ldots A_k$, respectively.

A transition of the register machine consists in updating the number stored in a register and in changing the current state to another one, according to an instruction. An increment instruction $(p, A+, q, s) \in P$ is performed if M is in state p, the number stored in register A is increased by 1, and after that M enters either state q or state s, chosen non-deterministically. A decrement instruction $(p, A-, q, s) \in P$ is performed if M is in state p, and if the number stored in register A is positive, then it is decreased by 1, and then M enters state q, and if the number stored in A is 0, then the contents of A remains unchanged and M enters state s.

We say that a register machine $M = (Q, R, q_0, q_f, P)$, with k registers, given as above, *generates* a non-negative integer n if starting from the *initial configuration* $(q_0, 0, 0, \ldots, 0)$ it enters the *final configuration* $(q_f, n, 0, \ldots, 0)$. The set of non-negative integers generated by M is denoted by $N(M)$.

For further details on register machines the reader is referred to [7]. It is known that they compute all recursively enumerable sets of non-negative integers; the family of these sets of numbers is denoted by NRE. We denote the family of finite sets of non-negative integers by $NFIN$.

Next we recall the basic definitions concerning generalized communicating P systems [14].

Definition 1. *A* generalized communicating P system *(a* GCPS*) of degree* n*, where* $n \geq 1$*, is an* $(n+4)$*-tuple* $\Pi = (O, E, w_1, \ldots, w_n, R, h)$ *where*

1. O *is an alphabet, called the* set of objects *of* Π*;*
2. $E \subseteq O$*; called the* set of environmental objects *of* Π*;*
3. $w_i \in O^*$*,* $1 \leq i \leq n$*, is the multiset of objects* initially associated to cell i*;*
4. R *is a finite set of* interaction rules *(or* communication rules*) of the form* $(a, i)(b, j) \to (a, k)(b, l)$*, where* $a, b \in O$*,* $0 \leq i, j, k, l \leq n$*, and if* $i = 0$ *and* $j = 0$*, then* $\{a, b\} \cap (O \setminus E) \neq \emptyset$*; i.e.,* $a \notin E$ *and/or* $b \notin E$*;*
5. $h \in \{1, \ldots, n\}$ *is the* output cell.

The system consists of n cells, labeled by natural numbers from 1 to n, which contain multisets of objects over O; initially cell i contains multiset w_i (the initial contents of cell i is w_i). We distinguish an additional special cell, labeled by 0, called the *environment*. The environment contains objects of E in an *infinite number of copies*.

The cells interact by means of the rules $(a, i)(b, j) \to (a, k)(b, l)$, with $a, b \in O$ and $0 \leq i, j, k, l \leq n$. As the result of the application of the rule, object a moves from cell i to cell k and b moves from cell j to cell l. If two objects from the environment move to some other cell or cells, then at least one of them must not appear in the environment in an infinite number of copies. Otherwise, an infinite number of objects can be imported in the system in one step.

A *configuration* of a GCPS Π, as above, is an $(n+1)$-tuple $(z_0, z_1, \ldots, z_n)$ with $z_0 \in (O \setminus E)^*$ and $z_i \in O^*$, for all $1 \leq i \leq n$; z_0 is the multiset of objects present

in the environment in a finite number of copies, whereas, for all $1 \leq i \leq n$, z_i is the multiset of objects present inside cell i. The *initial configuration of* Π is the tuple $(\lambda, w_1, \ldots, w_n)$.

Given a multiset of rules $\mathcal{R}$ over R and a configuration $u = (z_0, z_1, \ldots, z_n)$ of Π, we say that R is *applicable* to u if all its elements can be applied simultaneously to the objects of multisets $z_0, z_1, \ldots, z_n$ such that every object is used by at most one rule. Then, for a configuration $u = (z_0, z_1, \ldots, z_n)$ of Π, a new configuration $u' = (z'_0, z'_1, \ldots, z'_n)$ is obtained by applying the rules of R in a non-deterministic maximally parallel manner: taking an applicable multiset of rules $\mathcal{R}$ over R such that the application of $\mathcal{R}$ results in configuration $u' = (z'_0, z'_1, \ldots, z'_n)$ and there is no other applicable multiset of rules $\mathcal{R}'$ over R which properly contains $\mathcal{R}$.

One such application of a multiset of rules satisfying the conditions listed above represents a *transition* in Π from configuration u to configuration u'. A transition sequence is said to be a *successful generation* by Π if it starts with the initial configuration of Π and ends with a *halting configurations*, i.e., with a configuration where no further transition step can be performed.

We say that Π *generates a non-negative integer* n if there is a successful generation by Π such that n is the size of the multiset of objects present inside the output cell in the halting configuration. The *set of non-negative integers generated* by a GCPS Π in this way is denoted by $N(\Pi)$.

In the following we recall the notions of the possible restrictions on the interaction rules (modulo symmetry). Let O be an alphabet and let us consider an interaction rule $(a, i)(b, j) \to (a, k)(b, l)$ with $a, b \in O$, $i, j, k, l \geq 0$. Then we distinguish the following cases:

1. $i = j = k \neq l$: the *conditional-uniport-out rule*
   (the *uout* rule, for short) sends b to cell l provided that a and b are in cell i;
2. $i = k = l \neq j$: the *conditional-uniport-in rule*
   (the *uin* rule, for short) brings b to cell i provided that a is in that cell;
3. $i = j$, $k = l$, $i \neq k$: the *symport2 rule*
   (the *sym*2 rule, for short) corresponds to the minimal symport rule [10], i.e., a and b move together from cell i to k;
4. $i = l$, $j = k$, $i \neq j$: the *antiport1 rule*
   (the *anti*1 rule, for short) corresponds to the minimal antiport rule [10], i.e., a and b are exchanged in cells i and k;
5. $i = k$ and $i \neq j$, $i \neq l$, $j \neq l$: the *presence-move rule*
   (the *presence* rule, for short) moves the object b from cell j to l, provided that there is an object a in cell i and i, j, l are pairwise different cells;
6. $i = j$, $i \neq k$, $i \neq l$, $k \neq l$: the *split rule*
   (the *split* rule, for short) sends a and b from cell i to cells k and l, respectively;
7. $k = l$, $i \neq j$, $k \neq i$, $k \neq j$: the *join rule*
   (the *join* rule, for short) brings a and b together to cell i;
8. $i = l$, $i \neq j$, $i \neq k$ and $j \neq k$: the *chain rule*
   (the *chain* rule, for short) moves a from cell i to cell k while b is moved from cell j to cell i, i.e., to the cell where a was previously ;

9. i, j, k, l are pairwise different numbers: the *parallel-shift rule* (the $shift$ rule, for short) moves a and b from two different cells to another two different cells.

A generalized communicating P system may have rules of several types as defined above. When it only has one of them, then we call the corresponding GCPS a *minimal interaction P system* (with the given type of rules), or a GCPSMI, for short.

In the following, $NOtP_k(x)$ denotes the family of the sets of numbers generated by minimal interaction P systems of degree k and with rules of type x, $k \geq 1$ and $x \in \{uout, uin, sym2, anti1, presence, split, join, chain, shift\}$, and $NOtP_*(x)$ is the notation for $\bigcup_{k=1}^{\infty} NOtP_k(x)$. If the number of objects in the alphabet of objects in the GCPSMI is l, then the previous notations are changed to $NO_ltP_k(x)$ and $NO_ltP_*(x)$, respectively. We call these GCPSMIs *l-symbol minimal interaction P systems* (with a given type of rules).

Throughout the paper, we also refer to the symbol in the one-symbol P systems as token (denoted by $\bullet$). Furthermore, without any loss of the generality we may assume that one-symbol GCPSMIs are given over alphabet $O = \{\bullet\}$.

Notice that in the case of one-symbol minimal interaction P systems, the concepts of rule types conditional-uniport-out, symport2, antiport1, and split are not applicable. Since $O = E$, in these cases the rules $(\bullet, i)(\bullet, j) \to (\bullet, k)(\bullet, l)$ do not satisfy condition 4. of Definition 1, namely, that if $i = 0$ and $j = 0$, then $\{\bullet\} \cap (O \setminus E) \neq \emptyset$.

Due to their simplicity, the generative power of minimal interaction P systems is of particular interest. In [14,4] and it was shown that $NOtP_*(anti1) \subset NFIN$,

$$NRE = NOtP_{30}(uin) = NOtP_{30}(uot) = NOtP_9(split) = NOtP_7(join) = \\ = NOtP_{36}(presence) = NOtP_{19}(shift) = NOtP_*(chain),$$

and moreover, accepting systems with ten cells and $sym2$ rules are also able to characterize any set from NRE.

These results are valid if no restriction is introduced on the size of the object alphabet. In the following we restrict the size of the alphabet of objects to one, and will investigate the generative power of one-symbol minimal interaction P systems.

We observe that such systems are similar to Petri Nets having a restricted topology. This is especially visible if a graphical notation is used. However, the maximal parallelism and the concept of the environment are specific to P systems, so we place this study in the latter framework. A converse study of P systems from the point of view of Petri Nets can be found in [5]. For more details on Petri Nets and membrane computing we also refer to [10].

3 Main Results

In this section we show that one-symbol minimal interaction P systems with any of rule types parallel-shift, presence-move, chain, or join are able to generate every recursively enumerable set of numbers. In the case of systems Π with

conditional-uniport-in rules, either a finite set of non-negative numbers is generated, or there exists a $K \in \mathbb{N}$ such that $N(\Pi)$ contains any natural number $l \geq K$.

We first start with one-symbol minimal interaction P systems with parallel-shift rules and show their computational completeness. A similar result is given in Theorem 4.4 of [5]. The definition used there is slightly different, but the construction can be adapted to the case of GCPSMI with parallel-shift rules. We give below a different proof and use it in the rest of the sequel.

Theorem 1. $NO_1tP_*(shift) = NRE$.

Proof. Let $S \in NRE$ be generated by a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \ldots A_n\}$, $n \geq 1$ (M is given as in Section 2). We construct a one-symbol minimal interaction P system $\Pi = (O, E, w_1, \ldots, w_r, R_1, h)$, $r \geq 1$, with parallel-shift rules such that $N(M) = N(\Pi)$ holds. The proof idea is based on the simulation of the application of the instructions of M by applications of rule sets of Π.

The components of Π are defined as follows. Let $O = E = \{\bullet\}$, and let Π have for any $p \in Q$ cells labeled by elements of $\{p, 1_p, 2_p, 3_p, 4_p, 5_p\}$. (Note that for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q, s)$.) Furthermore, let Π have for any register A_i, $1 \leq i \leq n$, in M a dedicated cell labeled by A_i; the output cell is the one which corresponds to the output register of M. Let these cells be pairwise different and also different from the cells defined previously to the instructions. For the sake of simplicity, throughout the paper, we use terms "cell p" , "cell A_i" and "cell labeled by p", "cell labeled by A_i" as equivalent, respectively.

We define the rule set, R_1, of Π as follows. For any increment instruction (p, A_i+, q, s) of M, $1 \leq i \leq n$, R_1 contains the following rules.

$$(1^{(p)}) : (\bullet, p)(\bullet, 0) \to (\bullet, q)(\bullet, A_i) \qquad (2^{(p)}) : (\bullet, p)(\bullet, 0) \to (\bullet, s)(\bullet, A_i)$$

For any decrement instruction (p, A_i-, q, s) of M, $1 \leq i \leq n$, R_1 contains the following rules.

$$(1^{(p)}) : (\bullet, p)(\bullet, 0) \to (\bullet, 1_p)(\bullet, 2_p) \qquad (2^{(p)}) : (\bullet, 1_p)(\bullet, A_i) \to (\bullet, 3_p)(\bullet, 0)$$
$$(3^{(p)}) : (\bullet, 2_p)(\bullet, 0) \to (\bullet, 4_p)(\bullet, 5_p) \qquad (4^{(p)}) : (\bullet, 1_p)(\bullet, 4_p) \to (\bullet, s)(\bullet, 0)$$
$$(5^{(p)}) : (\bullet, 3_p)(\bullet, 4_p) \to (\bullet, q)(\bullet, 0)$$

Furthermore, R_1 consists only of the rules defined above.

The initial contents of the cells are given by $w_{q_0} = \bullet$, and $w_x = \lambda$ for $x \in Q \cup \{k_y \mid 1 \leq k \leq 5, y \in Q\} \cup \{A_i \mid 1 \leq i \leq n\}$, this configuration corresponds to the initial configuration of M.

We show that the application of an increment instruction or a decrement instruction of M can be simulated by the application of the corresponding rule set of Π defined above.

The reader may easily notice that the presence of a token ($\bullet$) in cell p means that the current state of M is p, therefore the rules $(1^{(p)})$ and $(2^{(p)})$ describe the application of an increment instruction of M.

Suppose now that the instruction of M to be simulated is (p, A_i-, q, s). Then cell p contains a token. (We note that no cell p, where $p \in Q$ has more than one token during any step of the generation). At the first step of the simulation, only rule $(1^{(p)})$ is applicable which moves the token from cell p and one token from the environment to cells 1_p and 2_p, respectively. If cell A_i contains at least one token, then, by rule $(2^{(p)})$, a token from cell 1_p moves to cell 3_p and one token from cell A_i exits to the environment. Meantime, the token from cell 2_p and one token from the environment are transported to cells 4_p and 5_p, respectively. At the next step, the token in cell 3_p moves to cell q and the token in cell 4_p exits to the environment, thus, the obtained configuration of Π corresponds to the configuration of M after performing the decrement instruction if register A_i contained at least one symbol. If cell A_i does not contain any token, then after performing rule $(1^{(p)})$, the only applicable rule is $(3^{(p)})$, which sends one-one tokens to cell 4_p and cell 5_p. After then, rule $(4^{(p)})$ is applied, which moves the token from cell 1_p to cell s and sends the token in cell 4_p to the environment.

Examining the proof above, the reader may observe that the generation process in Π is governed by the token arriving in a cell labeled by a state of M and no simulation of simultaneous instructions of M is possible.

By the construction of R_1, if M enters state q_f, then Π halts, since there is no applicable rule if cell q_f contains a token. It also can easily be seen that the number of tokens at the output cell is equal to the number stored in the output register of M by halting. Thus, $N(M) = N(\Pi)$.

Using in part the construction in the proof of Theorem 1, we show that one-symbol minimal interaction P systems with join rules are also computationally complete.

Theorem 2. $NO_1tP_*(join) = NRE$.

Proof. The proof, as the previous one, is based on simulation of the work of a register machine. Let $S \in N(RE)$ be generated by a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \ldots A_n\}$, $n \geq 1$. (M is given as in Section 2.) We now construct a one-symbol minimal interaction P system $\Pi = (O, E, w_1, \ldots, w_s, R_1, h)$, $s \geq 1$, with join rules such that $N(M) = N(\Pi)$ holds. The construction of Π is done in several steps.

We first note that by Theorem 1 there exists a one-symbol minimal interaction P system $\Pi' = (\{\bullet\}, \{\bullet\}, w'_1, \ldots, w'_r, R'_1, h')$, $r \geq 1$, with parallel-shift rules which generates S. Suppose that Π' is given as the GCPSMI in the proof of Theorem 1. It is easy to see that the application of any parallel-shift rule $(t) : (\bullet, i)(\bullet, j) \rightarrow (\bullet, k)(\bullet, l)$, where i, j, k, l are labels of cells and (t) is the label of the rule, can be simulated by the application of a join rule $(t') : (\bullet, i)(\bullet, j) \rightarrow (\bullet, c_t)(\bullet, c_t)$ followed by a split rule $(t'') : (\bullet, c_t)(\bullet, c_t) \rightarrow (\bullet, k)(\bullet, l)$, where c_t is a new cell introduced to rule $(t) : (\bullet, i)(\bullet, j) \rightarrow (\bullet, k)(\bullet, l)$. The new cells are pairwise different and different from the already existing ones. By this observation, we can construct a one-symbol minimal interaction P system Π'' such that $N(\Pi'') = N(\Pi')$ and the rule set of Π'' consists of the join and split rules constructed to the rules of Π' in the previously described manner. Then,

starting from Π'', we will construct Π. To do this, for any split rule in Π'' we design a set of rules in Π which rule set simulates the application of the split rule and only that. For this reason, we first define a block of cells, a so-called pseudo-split block. The term "pseudo-split" refers to that the rule set realizes a split if some special conditions hold.

A pseudo-split block is given as follows (see Figure 1). The block aims to split two tokens from cell 1 to cells 2 and 3. Cell 1 is supposed to have at least two tokens. One of them is sent to cell 2 and the other one to cell 3. Furthermore, one of the following conditions must hold:

(i) In one of the nodes 2 and 3 only one token can leave the cell to outside the block in the next step of the generation.
(ii) In one of the nodes 2 and 3 no token can leave the cell to outside the block for at least the next step of the generation.

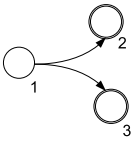


Fig. 1. A pseudo-split block

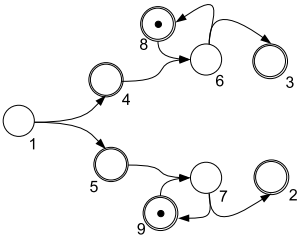


Fig. 2. Pseudo-split blocks simulating a split rule

The pseudo-split block is implemented by join rules as follows (initially cells $4, 4'$ have two tokens and cell $\#$ is a so-called trap cell):

$$(1) : (\bullet, 1)(\bullet, 4) \to (\bullet, 5)(\bullet, 5) \qquad (2) : (\bullet, 1)(\bullet, 4') \to (\bullet, 5')(\bullet, 5')$$
$$(3) : (\bullet, 4)(\bullet, 5) \to (\bullet, 3)(\bullet, 3) \qquad (4) : (\bullet, 4')(\bullet, 5') \to (\bullet, 2)(\bullet, 2)$$
$$(5) : (\bullet, 4)(\bullet, 5') \to (\bullet, \#)(\bullet, \#) \qquad (6) : (\bullet, 4')(\bullet, 5) \to (\bullet, \#)(\bullet, \#)$$
$$(7) : (\bullet, 5)(\bullet, 3) \to (\bullet, 4)(\bullet, 4) \qquad (8) : (\bullet, 5')(\bullet, 2) \to (\bullet, 4')(\bullet, 4')$$

The rules corresponding to the trap cell are the following:

$$(\#_1) : (\bullet, \#)(\bullet, 0) \to (\bullet, \bar{\#})(\bullet, \bar{\#}) \qquad (\#_2) : (\bullet, \bar{\#})(\bullet, 0) \to (\bullet, \#)(\bullet, \#)$$

We explain how the block functions. At starting, the only possibility not leading to an infinite generation is to equally distribute the tokens found at cell 1 between cells 5 and $5'$ (rules (1) and (2)). After that, two tokens will arrive at cell 2 and two tokens at cell 3 (from cells $4'$ and $5'$ and from cells 4 and 5, respectively). Suppose now that no token can leave cell 3 in the next step (i.e., condition (ii) holds). Then, one token from cell 3 and one token from cell 5 move to cell 4. This implies that at the same time one token from cell 2 and one token

from cell $5'$ will move to cell $4'$, because if this does not take place, then at the next step a token from cell 4 and the one remained in $5'$ will move to the trap cell. Since the construction is symmetric, a similar generation phase will be performed if it is cell 2 that cannot evolve for one step. The block functions similarly if condition (i) holds.

Notice that the pseudo-split block corresponds to a split rule only if at least one of the conditions (1) and (2) holds. In the following we construct a block arrangement consisting of pseudo-split blocks such that the above criterion is satisfied (see Figure 2).

The block arrangement functions as follows. Initially, each cell 8 and 9 contains one token. Firstly, a pseudo-split block is used to separate two tokens from cell 1 to cells 4 and 5. After then, a join rule is applied to send a token from cell 4 and cell 8 to cell 6 and from cell 5 and cell 9 to cell 7, respectively. This implies that the destination cells of the previously performed pseudo-split, i.e., cells 4 and 5 satisfy condition (i), since only one token can leave these cells. This phase of the generation is followed by two pseudo-splits, where the destination cells satisfy condition (ii), i.e., no token will be able to leave cell 8 and 9, respectively, at the next step. This implies that no simulation of another split instruction can start before the tokens coming from cell 1 arrive at cell 2 and 3. Hence, a split instruction is performed.

Combining join and split operations as described above, the rule set of Π can be constructed. We leave the details of the construction to the reader. It can easily be seen that Π generates the same set of numbers as Π', thus $N(\Pi) = N(M)$ holds.

Next we show that one-symbol minimal interaction P systems with presence-move rules are also computationally complete.

Theorem 3. $NO_1tP_*(presence) = NRE$.

Proof. Let $S \in NRE$ and let S be generated by a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \dots A_n\}$, $n \geq 1$. (M is given as in Section 2.) As in the case of the previous statements, we show that a one-symbol minimal interaction P system Π with presence-move rules can be constructed such that $N(M) = N(\Pi)$ holds. GCPSMI Π is defined in several steps.

Instead of direct simulations of the increment and decrement instructions of M, we define sets of rules, called (primitive) blocks, as it was done in [14,13,4] and then we show how a set of rules simulating the application of an increment instruction or that of a decrement instruction can be constructed from these blocks. We will use three types of blocks: the *uniport block*, the *main block*, and the *zero block*.

The *uniport block* is denoted by an arrow between circles labeled by i and j. It corresponds to the move of a token from cell i to cell j. This action is simulated by the following presence-move rule: (we suppose that a token is present initially in cell i'): $(\bullet, i')(\bullet, i) \to (\bullet, i')(\bullet, j)$.

The *main block*, see Figure 3, permits to move synchronously a token from cell i to cell j and a token from cell k to cell m. If no token is present in cell

k, then an infinite loop occurs. The arrows show the direction of the move of the objects and the circles corresponds to the cells. Since the semantics of the block is not symmetric, the double circle indicates the place of the symbol that triggers the generation and for which the infinite loop can occur.

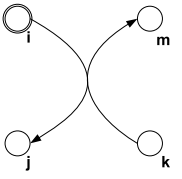


Fig. 3. The main block

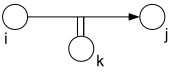


Fig. 4. The zero block

The *zero block*, see Figure 4, moves a token from cell i to cell j providing that there is no token in cell k. If this is not the case, then the generation enters an infinite loop. The notations are analogous to the ones used in Figure 3, namely, the arrow denotes the direction of the movement of the object, the circles denote cells, the double line and the circle labeled with k refer to the condition that no token is present in cell k.

In the following we show how the main block and the zero block are implemented in Π. The simulation of the main block is done by the following rule set. We suppose that initially each of cells 5, 8 and 13 contains a token.

$$(1) : (\bullet, 1)(\bullet, 5) \to (\bullet, 1)(\bullet, 6) \qquad (2) : (\bullet, 6)(\bullet, 1) \to (\bullet, 6)(\bullet, 7)$$
$$(3) : (\bullet, 7)(\bullet, 3) \to (\bullet, 7)(\bullet, 4) \qquad (4) : (\bullet, 8)(\bullet, 6) \to (\bullet, 8)(\bullet, 9)$$
$$(5) : (\bullet, 9)(\bullet, 7) \to (\bullet, 9)(\bullet, 12) \qquad (6) : (\bullet, 12)(\bullet, 9) \to (\bullet, 12)(\bullet, 5)$$
$$(7) : (\bullet, 5)(\bullet, 12) \to (\bullet, 5)(\bullet, 2) \qquad (8) : (\bullet, 8)(\bullet, 9) \to (\bullet, 8)(\bullet, 10)$$
$$(9) : (\bullet, 8)(\bullet, 10) \to (\bullet, 8)(\bullet, 11) \qquad (10) : (\bullet, 8)(\bullet, 11) \to (\bullet, 8)(\bullet, 10)$$
$$(11) : (\bullet, 13)(\bullet, 7) \to (\bullet, 13)(\bullet, 10)$$

These rules are also depicted on Figure 5 where the arrow represents the movement direction and the dashed line the controlling cell.

First, by applying rule (1), a token from cell 1 and the token from cell 5 (notice that cell 1 may contain more than one tokens) moves to cell 6. Then, the use of rule (4) will lead to an infinite generation, therefore rule (2) is used. At the next step, two possibilities may occur depending on whether cell 3 contains a token or not. Suppose that there exists a token in cell 3. Then rules (3) and (4) are applied in parallel, thus a token in cell 3 moves to cell 4 and the token in cell 6 moves to cell 9. After that, by rule (5), the token in cell 7 is transported to cell 12, and then, by rule (6), the token leaves cell 9 and arrives in cell 5. Then, the token in cell 12 moves to cell 2 (rule (7)), which means that the operations of the main block are performed, i.e., a token from cell 1 moved to cell 2 and a token from cell 3 moved to cell 4. Furthermore, the conditions of the initial configuration hold as well, i.e., each of cells 5, 8, 13 contains one token. If cell 3 does not

contain a token, then after performing rule (2), an infinite generation follows (the token moves infinitely many times between cells 10 and 11). The reader may easily see that the above rules can properly function only in the previously described manner. Notice that if we want to ensure that only one symbol in cell 1 is processed by this block, then we add rule (12):$(\bullet, 14)(\bullet, 1) \rightarrow (\bullet, 14)(\bullet, 10)$ to the above rule set.

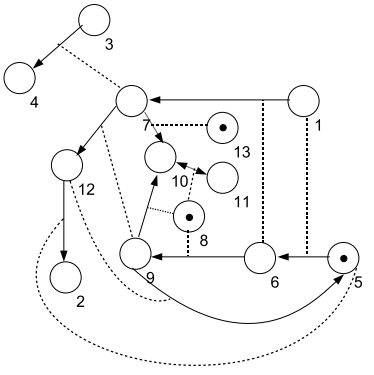


Fig. 5. A set of presence-move rules implementing a main block

The simulation of the zero block can be obtained from the rule set above, by eliminating rule (11) and replacing rule (3) : $(\bullet, 7)(\bullet, 3) \rightarrow (\bullet, 7)(\bullet, 4)$ with $(\bullet, 3)(\bullet, 7) \rightarrow (\bullet, 3)(\bullet, 10)$. In this case, if rules (3) and (4) are simultaneously applied, i.e., if there is a token in cell 3, then a token will appear in cell 10 leading to an infinite generation by rules (9) and (10). If rule (3) is not applicable, then applying the sequence of rules (4), (5), (6), (7), a token from cell 1 is successfully moved to cell 2 on the condition that cell 3 does not contain any token. The reader may notice that the rules cannot be applied in any other manner as described previously.

Now we construct block arrangements of rules of Π which simulate the increment instructions and the decrement instructions of M, illustrated by Figure 6 and Figure 7, respectively.

For any increment instruction (p, A_i+, q, s) of M, R_1 contains a rule set which is the implementation of a main block with the following modifications: cell 1 is replaced by cell p, cell 3 by the environment, and cell 4 by cell A_i. Cells i are replaced by cells $i^{(p)}$ for $5 \leq i \leq 12$, respectively. Since M may enter from state p either state q or state s, which both represent cell 2 in Π, therefore, instead of rule (7) : $(\bullet, 5)(\bullet, 12) \rightarrow (\bullet, 5)(\bullet, 2)$, we consider $(\bullet, 5^{(p)})(\bullet, 12^{(p)}) \rightarrow (\bullet, 5^{(p)})(\bullet, q)$ and $(\bullet, 5^{(p)})(\bullet, 12^{(p)}) \rightarrow (\bullet, 5^{(p)})(\bullet, s)$. The reader may immediately see that through the main blocks described above, any increment instruction of M can be simulated with Π.

For a decrement instruction (p, A_i-, q, s) of M, R_1 contains two uniport blocks $(\bullet, 1')(\bullet, p) \rightarrow (\bullet, 1')(\bullet, 2)$ and $(\bullet, 1')(\bullet, 2) \rightarrow (\bullet, 1')(\bullet, 3)$, i.e., we guess whether or not cell A_i contains a token. Furthermore, it contains a block arrangement

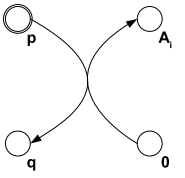


Fig. 6. Block arrangement for simulating an increment instruction

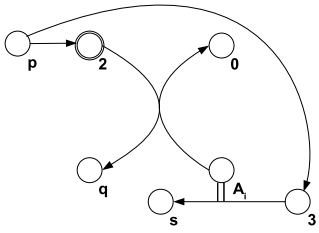


Fig. 7. Block arrangement for simulating a decrement instruction

which is a combination of a main block and a zero block. The main block moves a token from cell 2 to cell q and one token from cell A_i to the environment (if A_i contains at least one token and a token from cell p was sent to cell 2), or the token sent from cell p to cell 3 is forwarded to cell s (if A_i does not contain any token). In any other case, the constructed block arrangement implies the occurrence of an infinite loop. It is easy to see that the block arrangement described above simulates the application of the decrement instruction of M.

M halts in the final state q_f and the result of the generation is the number stored in its output register, A_h. To simulate the halting in M, we do not associate any rule to cells q_f, thus no further generation steps in Π can be performed.

At the beginning of the generation Π contains a token in cell q_0, $1'$, and in all the cells in the main blocks used for simulating the increment instructions and in the block arrangements used for simulating the decrement instructions which should initially contain at least one token. Due to the construction of R_1, the generation in Π is governed by cells p which correspond to the instructions of M to be performed, therefore any successful generation in Π corresponds to a successful generation in M and conversely. This implies that $N(M) = N(\Pi)$.

Over a one-symbol alphabet of objects, any chain rule corresponds to a presence-move rule, therefore the following statement is a direct consequence of Theorem 3.

Corollary 1. $NO_1tP_*(chain) = NRE$.

Next we describe the generative power of minimal interaction P systems with conditional-uniport-in rules.

Theorem 4. *For any GCPSMI Π with conditional-uniport-in rules either $N(\Pi)$ is finite or there is a $K \in \mathbb{N}$ such that $l \in N(\Pi)$ for every $l \geq K$.*

Proof. Let $\Pi = (O, E, w_1, \ldots, w_r, R_1, h)$, $r \geq 1$, be a one-symbol minimal interaction P system with conditional-uniport-in rules. Since these rules involve only two cells, to any configuration c of Π, we can assign a directed graph $G(c)$ where node i represents the cell i of Π, $1 \leq i \leq r$. The edges of G are determined by the rules of Π, i.e., if there is a rule of the form $p : (\bullet, i)(\bullet, j) \rightarrow (\bullet, i)(\bullet, i)$ in R_1, and cell i is not empty, then there is a directed edge from node j to node

representing i. We call this graph the *communication graph* of Π in configuration c. Note that if cell i contains no token, then, despite that $p \in R_1$, $G(c)$ has no edge from node j to node i. In this case we say that the edge from node j to node i has been broken.

In what follows, if no confusion arises, we use Π and graph G, in particular, the terms node/cell and edge/rule as equivalent. Without the loss of any generality, we also assume that at the initial configuration every cell contains at least one token (due to the forms of the rules, if the cell is empty, then it will remain empty during the generation).

A node k for which there is an edge (h, k) in the representing graph G of Π in some configuration c is called a *neighbor* of node h in c. We denote by $NBR(h, c)$ the set of all neighbors of a node h in configuration c and by $NBR(h)$ the set of all nodes which are neighbors of h during any generation. We also recall that a knot in a graph is a subset of nodes X such that for every edge (i, j), $i \in X$ it holds $j \in X$, i.e., it is not possible to leave X.

We call a path $p(c) = i_0, i_1, \ldots, i_{l_p}$ in the communication graph $G(c)$ corresponding to configuration c of Π an *alive path* (of length l_p) if i_0 denotes the environment, $i_{l_p} = h$, and a token $(\bullet)$ from the environment can move to cell h in l_p steps via the cells of $p(c)$ in the given order. This means that after l_p steps, Π may reach a configuration where any cell of the path contains at least two tokens, i.e., any cell i_{j+1} can import a token from cell i_j, $0 \leq j \leq l_p - 1$.

The reader may observe that if the following condition does not hold, then $L(\Pi) = \{0\}$ or $L(\Pi) = \emptyset$.

(1) There exists a generation which can reach a configuration c with no set of nodes $X \subseteq (NBR(h, c) \cup NBR(i_0, c) \setminus \{i_1\})$ such that X forms a knot.

If there is always an X as above, such that it is a set of neighbors of the environment not containing i_1 (the first node from the alive path $p(c)$), that is, if $X \subseteq (NBR(i_0, c) \setminus \{i_1\})$, then $L(\Pi) = \emptyset$, otherwise, if X is different, then $L(\Pi) = \{0\}$.

Now we show that the existence of such an alive path implies that there exists a constant $K \in \mathbb{N}$ such that $l \in L(\Pi)$ for any $l \geq K$. Suppose that there exists a configuration c and an alive path $p(c)$ in Π which satisfies condition (1). Then there is a generation in Π which permits to bring any number of new tokens in the output cell as follows: a token that was imported at the first time from the environment by cell i_1 (at the first configuration change in cell i_1) at the next step will be used for bringing in one other token from the environment. Then, after $|p|$ steps the token that first entered from the environment arrives at cell h and at the same time each internal cell in the path, i.e., cells $i_1, \ldots, i_{l_p - 1}$ will contain at least two tokens.

Now we should prove that after some point the process can stop at any time. Our assertion is based on the following observation. If there are two cells i (with n tokens) and j with one token, and there is a rule $(\bullet, j)(\bullet, i) \rightarrow (\bullet, j)(\bullet, j)$ (i.e. an edge (i, j)), then by using all tokens present in j at every step, after at most n steps all tokens in cell i can be transported to cell j. This observation comes from

the fact that there is a generation, such that the number of tokens in cell j after k steps is 2^k, while the number of tokens in cell i can be at most $2^k(n-k)$. Using this procedure, it is possible to break an edge in the graph $G(c)$ representing Π in some configuration c, i.e., to obtain a configuration where the communication graph has not this edge anymore.

This observation implies that for an alive path $p(c) = i_0 \dots i_{l_p} = h$ of Π, it is possible to break all edges in the graph which are different from the edges (i_j, i_{j+1}), $1 \leq j \leq l_p - 1$. (We also break the edges of all other possible alive paths: edge by edge, starting from the one linked to the environment.) After that, using a similar procedure for every node in $p(c)$, we may obtain a configuration that all cells belonging to the path, except i_0 and h, contain exactly two tokens.

Finally, because condition (1) holds, there is no subset of neighbor nodes of h which forms a knot, for any configuration c and every node $k \in NBR(h, c)$, either there is a node $k' \in NBR(k, c)$ but $k' \notin NBR(h, c)$, or there is a path from k consisting of neighbors of h to some node $k' \in NBR(h, c)$ such that it has a neighbor $k'' \in NBR(k', c)$ which is not neighbor of h ($k'' \notin NBR(h, c)$). We observe that in the first case node k can be emptied, thus the edge (h, k) can be broken, while in the second, case all tokens can move from k to k' (for all possible k) which reduces to the first case.

In a similar way, since condition (1) holds, it is possible to break all edges (i_0, k), $k \in (NBR(i_0, c) \setminus \{i_1\})$. Then, it is possible to break at the same time the edges going out from h as well as edges going out from the environment (i_0).

Hence, we can obtain a configuration c, such that there is only one alive path $p(c)$, where each cell contains exactly two tokens. Moreover, in this configuration h has no neighbor, and the only neighbor of the environment is i_1. Let $K - 1$ be the number of tokens at cell h at this configuration. Take any $l \geq K$. In order to obtain l it is enough to perform $l - K + 1$ generation steps where every cell i_k belonging to p brings one token from i_{k-1} and sends one token to i_{k+1} and after that bring two tokens to cell h, thus breaking the last edge. Thus, h will contain l tokens. The generation will stop some steps later as follows. Firstly, the path $p(c)$ is broken edge by edge, starting from the edge linking to the environment. This concludes the proof.

4 Conclusions

In this paper we provided alternatives of register machines, i.e., computational complete computing devices over the simplest alphabet, but with other types of simple architectures and rules. Since the concept of four types of the restricted communication rules were not applicable for one-symbol minimal interaction P systems, but GCPSMIs with three of them were computational complete if no restriction was put on the size of the object alphabet, the problem of finding the minimal number k such that GCPSMIs with a given type of rule (except antiport1) with object alphabet of size at most k are computationally complete is of particular interest. Finally, another important topic for future research is the relation between one-symbol minimal interaction P systems and Petri nets. Similar questions have already been studied in [3,5].

References

1. Alhazov, A., Freund, R., Oswald, M., Verlan, S.: Partial versus total halting in P systems. In: Gutiérrez-Naranjo, M.A., et al. (eds.) Proceedings of the Fifth Brainstorming Week on Membrane Computing, Sevilla, pp. 1–20 (2007)
2. Alhazov, A., Freund, R., Rogozhin, Y.: Computational power of symport/antiport: History, advances, and open problems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 1–30. Springer, Heidelberg (2006)
3. Bernardini, F., Gheorghe, M., Margenstern, M., Verlan, S.: Producer/consumer in membrane systems and Petri nets. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 43–52. Springer, Heidelberg (2008)
4. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. Theoretical Computer Science (to appear)
5. Frisco, P.: Computing with Cells. Oxford University Press, Oxford (2009)
6. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
7. Minsky, M.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)
8. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Computing 20, 295–305 (2002)
9. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
10. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1-3. Springer, Heidelberg (1997)
12. The P systems webpage, `http://ppage.psystems.eu`
13. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: On communication in tissue P systems: conditional uniport. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 521–535. Springer, Heidelberg (2006)
14. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: Generalized communicating P systems. Theoretical Computer Science 404(1-2), 170–184 (2008)

A Faster P Solution for the Byzantine Agreement Problem

Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
{mjd,yun,radu}@cs.auckland.ac.nz

Abstract. We propose an improved generic version of P modules, an extensible framework for recursive composition of P systems. We further provide a revised P solution for the Byzantine agreement problem, based on Exponential Information Gathering (EIG) trees, for N processes connected in a complete graph. Each process is modelled by the combination of $N + 1$ modules: one "main" module, plus one "firewall" communication module for each process (including one for itself). The EIG tree evaluation functionality is localized into a "main" single cell P module. The messaging functionality is localized into a three cells communication P module. This revised P solution improves overall running time from $9L+6$ to $6L+1$, where L is the number of messaging rounds. Most of the running time, $5L$ steps, is spent on the communication overhead. We briefly discuss if single cells can solve the Byzantine agreement without support and protection from additional communication cells; we conjecture that this is not possible, within the currently accepted definitions.

Keywords: P systems, P modules, Byzantine agreement, Distributed algorithms, Modular design.

1 Introduction

Large distributed systems are typically *composed* from smaller building blocks. However, until recently, classical P systems did not offer enough support for effective programmability. Recent papers, such as [19,18,16], have started to address these problems. Guided by similar goals, we recently proposed a new modular framework, called *P modules*, that supports *generic objects*, *encapsulation*, *information hiding* and *recursive composition* [8]. Our proposal is compatible with any data structure based on directed arcs, i.e. it covers cell-like P systems (based on trees), hP systems (based on dags) and nP systems (based on digraphs).

In this paper, we extend this previous proposal, with *external definitions* and *external references*, which support safer and more flexible module interconnection facilities. We demonstrate its enhanced expressibility on a couple of simple examples, then we use it to provide a new and improved P systems solution to the Byzantine agreement problem.

The Byzantine agreement problem was first proposed by Pease *et al.* in 1980 [17] and further elaborated in Lamport *et al.*'s seminal paper [10]. This problem addresses a fundamental issue in complex systems: correctly functioning processes must be able to

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 175–197, 2010.

overcome their possible differences and achieve a consensus, despite arbitrarily faulty processes that can give conflicting information to different parts of the system.

The Byzantine agreement has become one of the most studied problems in distributed computing—some even consider it the "crown jewel" of distributed computing. Lynch covers many versions of this problem and their solutions, including a complete description of the classical algorithm, based on Exponential Information Gathering (EIG) trees as a data structure [11].

Recent years have seen revived interest in this problem and its solutions, to achieve higher performance or stronger resilience, in a wide variety of contexts [4,1,3,12], including, for example, solutions for quantum computers [2].

To the best of our knowledge, except our previous work on Byzantine agreement problem [8], no other complete solution for P systems has been published. In the context of P systems, this problem was briefly mentioned, without solutions [6,5]. Our solution was based on the classical EIG-based algorithm, where each EIG node was implemented by a distinct cell.

In this paper, we provide a revised P solution for the Byzantine agreement problem, based on EIG trees, for N processes connected in a complete graph. Each process is modelled by the combination of $N + 1$ modules: one "main" module, plus one "firewall" communication module for each process (including one for itself). The EIG tree evaluation functionality is localized into a "main" single cell P module. The messaging functionality is localized into a communication P module with three cells. This revised P solution uses only duplex channels, uses fewer cells and rules, and improves overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.

The rest of the paper is organized as follows. Section 2 covers a few basic preliminaries, then introduces a combinatorial definition of the EIG data structure. We describe the Byzantine agreement problem in detail in Section 3, which also includes a small case study with four processes. An extended version of P modules is formally introduced in Section 4. In Section 5, using our new modular framework, we model and develop the structure of a P systems implementation of the Byzantine agreement problem. The rules used in our design are described in Section 6, where we also discuss the correctness of our design. Finally, in Section 7, we summarize our results and discuss related open problems.

2 Preliminaries

We assume that the reader is familiar with the basic terminology and notations: functions, relations, graphs, nodes (vertices), arcs, directed graphs, dags, trees, alphabets, strings and multisets [13]. Given two sets, A, B, a subset f of their cartesian product, $f \subseteq A \times B$, is a *functional relation* if $\forall(x, y_1), (x, y_2) \in f \Rightarrow y_1 = y_2$. Obviously, any function $f : A \to B$ can be viewed a functional relation, $\{(x, f(x)) \mid x \in A\}$, and, vice-versa, any functional relation can be viewed as a function.

We now recall a few basic concepts from combinatorial enumerations. The *integer range* from m to n is denoted by $[m, n]$, i.e. $[m, n] = \{m, m+1, \ldots, n\}$, if $m \leq n$, and $[m, n] = \emptyset$, if $m > n$. The set of *permutations* of n of length m is denoted by $P(n, m)$, i.e. $P(n, m) = \{\pi : [1, m] \to [1, n] \mid \pi \text{ is injective}\}$. A permutation π is represented

by the sequence of its values, i.e. $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, and we will often abbreviate this further as the sequence $\pi = \pi_1.\pi_2 \ldots \pi_m$. The sole element of $P(n, 0)$ is denoted by $()$, or by λ, if the context removes any possible ambiguity. Given a subrange $[p, q]$ of $[1, m]$, we define a *subpermutation* $\pi(p : q) \in P(n, q - p + 1)$ by $\pi(p : q) = (\pi_p, \pi_{p+1}, \ldots, \pi_q)$. The *image* of a permutation π, denoted by $Im(\pi)$, is the set of its values, i.e. $Im(\pi) = \{\pi_1, \pi_2, \ldots, \pi_m\}$. The *concatenation* of two permutations is denoted by $\oplus$, i.e. given $\pi \in P(n, m)$ and $\tau \in P(n, k)$, such that $Im(\pi) \cap Im(\tau) = \emptyset$, $\pi \oplus \tau = (\pi_1, \pi_2, \ldots, \pi_m, \tau_1, \tau_2, \ldots, \tau_k) \in P(n, m + k)$.

An *Exponential Information Gathering* (EIG) tree $T_{N,L}$, $N \geq L \geq 0$, is a labelled (ordered) rooted tree of height L that is defined recursively as follows. The tree $T_{N,0}$ is a rooted tree with just one node, its root, labelled λ. For $L \geq 1$, $T_{N,L}$ is a rooted tree with $1 + N|T_{N-1,L-1}|$ nodes (where $|T|$ is the size of tree T), root λ, having N subtrees, where each subtree is isomorphic with $T_{N-1,L-1}$ and each node, except the root, is labelled by the least element of $[1, N]$ that is different from any ancestor node or any left sibling node. Alternatively, $T_{N,L-1}$ is isomorphic and identically labelled with the tree obtained from $T_{N,L}$ by deleting all its leaves. It is straightforward to see that there is a bijective correspondence between the permutations of $P(N, L)$ and the sequences (concatenations) of labels on all paths from root to the leaves of $T_{N,L}$. Thus, each node σ in an EIG tree $T_{N,L}$ is uniquely identified by a permutation $\pi_\sigma \in P(N, l)$, where $l \in [0, L]$ is also σ's depth, and, vice-versa, each such permutation π has a corresponding node σ_π. We will further use this node-permutation identification, while referring to nodes.

Given EIG tree $T_{N,L}$, an attribute is a function $\aleph : T_{N,L} \to V$, for some value set V; alternatively, $\aleph$ can be given as a functional subset of $\{\pi \in P(N, t) \mid t \in [0, L]\} \times V$.

See Figure 1 for an example of the EIG tree, $T_{4,2}$. Level 0 corresponds to permutation set $\{\lambda\}$. Level 1 corresponds to permutation set $\{(1), (2), (3), (4)\}$. Level 2 corresponds to permutation set $\{(1,2), (1,3), (1,4), (2,1), (2,3), (2,4), (3,1), (3,2), (3,4), (4,1), (4,2), (4,3)\}$. This tree is decorated with two attributes, α and β. Using an alternate notation for permutations (to avoid embedded parentheses), attribute α corresponds to the functional relation $\{(\lambda, 1), (1, 0), (2, 0), (3, 1), (4, 1), (1.2, 0), (1.3, 0), (1.4, 1), (2.1, 0), (2.3, 0), (2.4, 0), (3.1, 0), (3.2, 1), (3.4, 1), (4.1, 1), (4.2, 1), (4.3, 1)\}$.

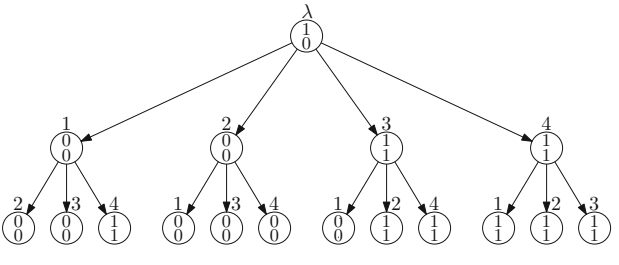


Fig. 1. A sample EIG tree, $T^3_{4,2}$, completed with two attributes, α and β. The node labels appear besides the node blob. Each node blob contains its two attribute values: the α value at the top, and the β value at the bottom.

3 The EIG-Based Byzantine Agreement Algorithm

Each process starts with its own initial decision choice (typically different). At the end, all non-faulty processes must take the same final decision, even if the faulty processes attempt to disrupt the agreement, accidentally or intentionally.

The classical EIG-based algorithm solves the Byzantine agreement problem in the *binary decision* case ($no = 0, yes = 1$), for N *processes*, connected in a *complete graph* (where edges indicate *reliable duplex communication lines*), provided that $N \geq 3F + 1$, where F is the maximum number of faulty processes. This is a *synchronous* algorithm; celebrated results (see for example [11]) show that the Byzantine agreement is *not* possible if $N \leq 3F$, in the *asynchronous* case or when the communication links are *not* reliable.

Without providing a complete description, we provide a sketch of the classical algorithm, *reformulated* on the basis of the theoretical framework introduced in Section 2. For a more complete and verbose description of this algorithm, including correctness and complexity proofs, we refer the reader to Lynch [11].

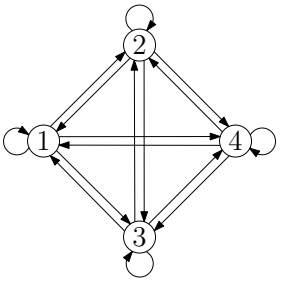
Each non-faulty process, h, has its own copy of an EIG tree, $T^h_{N,L}$, where $L = F+1$. This tree is decorated with two attributes, $\alpha^h, \beta^h : \{\pi \in P(N,t) \mid t \in [0,L]\} \to \{0, 1, \texttt{null}\}$, where `null` designates undefined items (not yet evaluated). Attributes α^h and β^h are also known as val_h and $newval_h$ [11], or *top-down* and *bottom-up* [8]. As their alternative names suggest, α^h is first evaluated, in a top-down tree traversal, in increasing level order; next, β^h is evaluated, in a bottom-up traversal, in decreasing level order.

The algorithm works in two phases. Its *first phase* is a *messaging* phase which completes the evaluation of the top-down attribute α^h. Initially, $\alpha^h(\lambda) = v^h$, the initial choice of process h; all the other α^h and β^h values are still undefined. Next, there are L messaging rounds. At round $t \in [1, L]$, h broadcasts to all processes (including self), a reversibly encoded message which identifies its α^h values at level $t-1$, i.e. the set $\{(\pi, \alpha^h(\pi)) \mid \pi \in P(N, t-1)\}$. All other non-faulty processes broadcast messages, in a similar way. Process h decodes and processes the messages that it receives. From process f, $f \in [1, N]$, process h receives the set $\{(\pi, \alpha^f(\pi)) \mid \pi \in P(N, t-1)\}$. Each item $(\pi, \alpha^f(\pi))$, where $f \notin Im(\pi)$, is used to assign further α^h values, to the next level down the EIG tree, by setting $\alpha^h(\pi \oplus f) = \alpha^f(\pi)$; items where $f \in Im(\pi)$ are silently discarded. As this formula suggests, it is indeed *critical* that h "knows" the origin f of each received message and that this origin mark cannot be faked by faulty processes. Wrong or missing values are replaced by the value of a predefined default parameter, $W \in \{0, 1\}$. Thus, there are L messaging rounds and, after the last round, all nodes are decorated with values of attribute α. In fact, only the last level α values are actually needed, to start the next phase, a practical implementation can choose to discard the other α values.

Next, the algorithm switches to its *second phase*, the evaluation of the bottom-up attribute β^h. First, for leaves, $\beta^h(\pi) = \alpha^h(\pi), \pi \in P(N, L)$. Next, given β^h values for level $t \in [1, L]$, each β^h value for the next level up, $\beta^h(\pi), \pi \in P(N, t-1)$, is evaluated on the basis of the β^h values of node π's children, i.e. on the multiset $\{\beta^h(\pi \oplus f) \mid f \in [1, N] \setminus Im(\pi)\}$, using a local majority voting scheme: $\beta^h(\pi) = 0$, if a strict majority of the above multiset values are 0; or, $\beta^h(\pi) = 1$, if a strict majority of

the above multiset values are 1; or, $\beta^h(\pi) = W$ (the same default parameter mentioned above), if there is a tie. At the end, the β^h value for the EIG root, $\beta^h(\lambda)$, is process h's final decision. All non-faulty processes will simultaneously reach the same final decision; any decision taken by faulty nodes is not relevant.

Example 1 (Sample Byzantine scenario). Consider a Byzantine scenario with $N = 4$ and $F = 1$, thus $L = 2$. Assume that processes 1, 2, 3 and 4 start with initial choices 0, 0, 1, and 1, respectively. Further, assume that process 1 is faulty and these four processes exchange the messages described in Figure 2. For a more verbose description of this example, please see our technical paper [7].



Process	1	2	3	4
Initial choice	0	0	1	1
Faulty	Yes	No	No	No
Round 1 messages	(λ, x)	$(\lambda, 0)$	$(\lambda, 1)$	$(\lambda, 1)$
Round 2 messages	$(1,0)(2,0)$ $(3,y)(4,1)$	$(1,0)(2,0)$ $(3,1)(4,1)$	$(1,0)(2,0)$ $(3,1)(4,1)$	$(1,1)(2,0)$ $(3,1)(4,1)$
Final decision	?	0	0	0

Fig. 2. A sample Byzantine scenario, $N = 4$, $F = 1$, where process 1 is faulty

4 P Modules

For this section, and the rest of the paper, we assume familiarity with P systems [14,15], nP systems [15] and hP systems [13]. We will also use a terminology inspired from standard modular design.

Intuitively, a P module is one of the above P systems, with additional features, required for its further assembly into a larger P module. A P module exposes the following additional features, collectively called *generic parameters*, which can be further *instantiated*, when the current module is *combined* with other modules:

- Besides objects of the initial alphabet, rules can also use *generic symbols*. A generic symbol, abbreviated as `sym`, can be instantiated to one of the already existing objects or as a new object (thereby extending the initial alphabet).
- Cells can be designated as *external definitions*. An external definition indicates either the start or the end of a potential arc and is abbreviated as $\texttt{def}_{\uparrow}$ or as $\texttt{def}_{\downarrow}$, respectively.
- Arcs can be designated as *external arcs*, indicating potential arcs between existing cells and external definitions of other modules. An external arc has one uninstantiated start or end cell, called an *external reference*, which is abbreviated as $\texttt{ref}_{\uparrow}$ or as $\texttt{ref}_{\downarrow}$, respectively.
- An *external arc* can be instantiated by identifying its external reference to a matching external definition from another module, i.e. either its $\texttt{ref}_{\downarrow}$ reference to a $\texttt{def}_{\downarrow}$ definition, or its $\texttt{ref}_{\uparrow}$ reference to a $\texttt{def}_{\uparrow}$ definition.

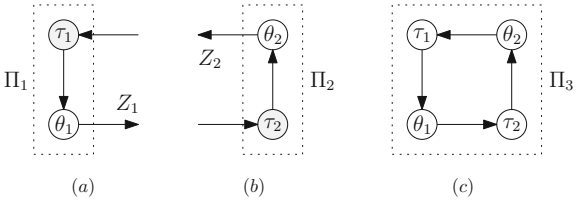


Fig. 3. Composing two simple P modules. Module Π_3 is the composition of modules Π_1 and Π_2, after instantiating $Z_1 = \tau_2$, $Z_2 = \tau_1$. External references are indicated by labels on outgoing arcs, Z_1 and Z_2, and external definitions by shaded cells, τ_1 and τ_2.

Figure 3 illustrates these intuitive ideas. Module Π_1 offers a $\texttt{def}_\downarrow$ definition, τ_1, and uses a $\texttt{ref}_\downarrow$ reference, Z_1. Module Π_2, which can be viewed as a copy of Π_1, offers a $\texttt{def}_\downarrow$ definition, τ_2, and uses a $\texttt{ref}_\downarrow$ reference, Z_2. Module Π_3 is the result of their composition, after instantiating $Z_1 = \tau_2$ and $Z_2 = \tau_1$, thereby instantiating arcs (θ_1, τ_2) and (θ_2, τ_1).

Definition 1 (P module). *A* P module *is a system* $\Pi = (O, K, \delta, S, D_\uparrow, D_\downarrow, R_\uparrow, R_\downarrow)$, *where:*

1. O *is a finite non-empty alphabet of* objects*;*
2. K *is a finite set of* cells*;*
3. δ *is a subset of* $(K \times K) \cup (K \times R_\downarrow) \cup (R_\uparrow \times K)$*, i.e. a set of parent-child structural arcs, representing* duplex *or* simplex *communication channels, between two existing cells or between an existing cell and an external reference;*
4. S *is a finite alphabet, disjoint of* O*, of generic* $\texttt{sym}$ *objects;*
5. $D_\uparrow$ *is a subset of* K*, representing* $\texttt{def}_\uparrow$ *definitions;*
6. $D_\downarrow$ *is a subset of* K*, representing* $\texttt{def}_\downarrow$ *definitions;*
7. $R_\uparrow$ *is a finite set, disjoint of* K*, representing* $\texttt{ref}_\uparrow$ *references;*
8. $R_\downarrow$ *is a finite set, disjoint of* K*, representing* $\texttt{ref}_\downarrow$ *references.*

Let $\bar{O} = O \cup S$ *be the original alphabet extended with the generic symbols. Each cell,* $\sigma \in K$*, has the form* $\sigma = (Q, s_0, w_0, R)$*, where:*

- Q *is a finite set of* states*;*
- $s_0 \in Q$ *is the* initial state*;*
- $w_0 \in \bar{O}^*$ *is the* initial multiset *of objects;*
- R *is a finite* ordered *set of multiset rewriting* rules *of the general form:* $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma}$*, where* $s, s' \in Q$*,* $x, x' \in \bar{O}^*$*,* $u \in \bar{O}^*$*,* $\alpha \in \{\texttt{min}, \texttt{max}\}$*,* $\beta \in \{\uparrow, \downarrow, \updownarrow, \leftrightarrow\}$*,* $\gamma \in \{\texttt{one}, \texttt{spread}, \texttt{repl}\} \cup K \cup R_\uparrow \cup R_\downarrow$*. If* $u = \lambda$*, this rule can be abbreviated as* $s\ x \rightarrow_\alpha s'\ x'$*. The semantics of the rules and of the* α*,* β*,* γ *operators are further described in the rest of this section.*

Remark 1. This definition of P module subsumes several earlier definitions of P systems, hP systems and nP systems. If δ is a *tree*, then Π is essentially a tree-based P system (which can also be interpreted as a cell-like P system). If δ is a *dag*, then Π is essentially an hP system. If δ is a *digraph*, then Π is essentially an nP system.

Remark 2. Most often, our P systems are introduced semi-formally, where the objects, cells, arcs and rules are inferred from diagrams and listings. In this case, we use angular brackets to emphasize generic parameters, together with their type. For example, the generic parameters of module Π_1 of Figure 3 can be indicated as $\langle \texttt{def}_{\downarrow}\ \tau_1, \texttt{ref}_{\downarrow}\ \theta_1 \rangle$, and the whole module can be emphasized as $\Pi_1 \langle \texttt{def}_{\downarrow}\ \tau_1, \texttt{ref}_{\downarrow}\ \theta_1 \rangle$.

The rules given by the ordered set R are attempted in *weak priority* order [15]. If a rule is *applicable*, then it is *applied* and then the next rule is attempted (if any). If a rule is not applicable, then the next rule is attempted (if any). Note that state-based rules introduce an extra requirement for determining rule applicability, namely the target state indicated on the right-hand side must be the same as the previously chosen target state (if any) [14,13]. Rules are applied under the usual immediate ("eager") evaluation of their left-hand sides and deferred ("lazy") evaluation of their right-hand sides [14].

With these conventions, one cell's ordered set of rules becomes a sequence of programming statements for a hypothetical P machine, where each rule includes a simple if-then-fi conditional test for applicability and, as we see below, some while-do-od looping facilities (`max` and `repl` operators), with some potential for in-cell parallelism, in addition to the more obvious inter-cell parallelism. State compatibility introduces another intra-cell if-then-fi conditional test, this time between rules.

The *rewriting* operator $\alpha = \texttt{min}$ indicates that the rewriting is applied once, if the rule is applicable; and $\alpha = \texttt{max}$ indicates that the rewriting is applied as many times as possible, if the rule is applicable.

The *transfer* operator $\beta = \uparrow$ indicates that the multiset u is sent "up", to the parents; $\beta = \downarrow$ indicates that the multiset u is sent "down", to the children; $\beta = \updownarrow$ indicates that the multiset u is sent both "up" and "down", to both parents and children; and $\beta = \leftrightarrow$, indicates "lateral" transfer, to the siblings (this $\leftrightarrow$ operator is not used in this paper).

The additional transfer operator $\gamma = \texttt{one}$ indicates that the multiset u is sent to one recipient (parent or child, according to the direction indicated by β). The operator $\gamma = \texttt{spread}$ indicates that the multiset u is spread among an arbitrary number of recipients (parents, children or parents and children, according to the direction indicated by β). The operator $\gamma = \texttt{repl}$ indicates that the multiset u is replicated and broadcast to all recipients (parents, children or parents and children, according to the direction indicated by β). The operator $\gamma = \sigma \in K \cup R_{\uparrow} \cup R_{\downarrow}$ indicates that the multiset u is sent to σ, if cell σ is in the direction indicated by β; otherwise, the multiset u is "lost". By convention, if cells have unique indices or are labelled and labels are locally unique, we can abbreviate $\gamma = \sigma$ by $\gamma = i$, where i is the index or label of σ.

The following examples illustrate the behaviour of these operators. Consider a cell σ, in state s and containing aa. Consider the potential application of a rule $s\ a \rightarrow_{\alpha} s'\ b\ (c)_{\beta_{\gamma}}$, by looking at specific values for α, β, γ operators:

- The rule $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied and, after its application, cell σ will contain ab and a copy of c will be sent to each of σ's parents.
- The rule $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied and, after being applied twice, cell σ will contain bb and a copy of cc will be sent to each of σ's parents.
- The rule $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$), can be applied and, after its application, cell σ will contain ab and a copy of c will be sent to σ', if σ' appears among the children of σ, otherwise, this c will be lost.

– The rule $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied and, after being applied twice, cell σ will contain bb and a copy of cc will be sent to σ', if σ' appears among the children of σ, otherwise, this cc will be lost.

In this paper, we are only interested in *deterministic solutions*, and we will exclusively use the `min`, `max`, `repl`, and K operators, and avoid operators with a higher potential for non-determinism, such as `par`, `one`, `spread`.

By default, the channels are *duplex*, allowing simultaneous transmissions from both ends. Although we do not use them here, *simplex* channels are also available in our model; a simplex channel indicates a single open direction, either from parent to child, or from child to parent (thus there is no necessary relation between the structural directions and communication direction); messages sent in the other direction are silently "lost".

Given an arbitrary finite set of P modules, we can construct a higher level P module by instantiating some of their external references to some of their external definitions, which implicitly instantiates some new arcs, and by instantiating some of their unspecified symbols. This construction requires that the original P modules are disjoint, in the sense specified below.

Consider a finite family of n P modules, $\mathcal{P} = \{\Pi_i \mid i \in [1,n]\}$, where $\Pi_i = (O_i, K_i, \delta_i, S_i, D_{\uparrow i}, D_{\downarrow i}, R_{\uparrow i}, R_{\downarrow i})$, $i \in [1,n]$. This family $\mathcal{P}$ is *cell-disjoint*, if their cell sets disjoint, i.e. $K_i \cap K_j = \emptyset$, for $i, j \in [1,n]$. If required, any such family can be made cell-disjoint, by a *deep copy* process, which clones all cells and, as a convenience, automatically allocates successive indices to cloned cells (e.g., starting from cell σ, the first cloned cell is σ_1, the second is σ_2, etc). However, a good practice is to systematically index all cells of a P module, by labels related to the generic parameters, such that distinct copies of the same generic module are automatically cell-disjoint. We will generally follow this convention.

Given a family $\mathcal{P}$, the result of a composition depends on the actual instantiations, i.e. which unspecified symbols are instantiated and which external references and definitions are matched. Symbol instantiation is specified by a partial mapping $\omega : \bigcup_{i\in[1,n]} S_i \rightarrow \Omega$, where Ω is a universal alphabet, covering all alphabets used in a given application. The symbols that have been instantiated are defined by the domain of ω, i.e. $\mathrm{Dom}(\omega)$, and their assigned objects by the image of ω, i.e. $\mathrm{Im}(\omega)$. External references are similarly matched to external definitions by two partial mappings, $\rho_\uparrow : \bigcup_{i\in[1,n]} R_{\uparrow i} \rightarrow \bigcup_{i\in[1,n]} D_{\uparrow i}$, $\rho_\downarrow : \bigcup_{i\in[1,n]} R_{\downarrow i} \rightarrow \bigcup_{i\in[1,n]} D_{\downarrow i}$. A previously uninstantiated arc (σ, x), $\sigma \in K_i$, $x \in R_{\downarrow i}$, $i \in [1,n]$, is instantiated as $(\sigma, \rho_\downarrow(x))$, and a previously uninstantiated arc (x, σ), $\sigma \in K_i$, $x \in R_{\uparrow i}$, $i \in [1,n]$, is instantiated as $(\rho_\uparrow(x), \sigma)$.

Definition 2 (P modules composition). *The P module $\Psi = (O, K, \delta, S, D_\uparrow, D_\downarrow, R_\uparrow, R_\downarrow)$ is a* composition *of the P module family $\mathcal{P}$, if:*

– *$\mathcal{P}$ is cell-disjoint;*
– *ω, $\rho_\uparrow$, $\rho_\downarrow$ are the partial mappings which define the instantiation (as previously introduced);*
– $O = \bigcup_{i\in[1,n]} O_i \cup \mathrm{Im}(\omega)$*;*
– $K = \bigcup_{i\in[1,n]} K_i$*;*

- $\delta = \{(\hat{\rho}_\uparrow(\sigma), \hat{\rho}_\downarrow(\sigma)) \mid (\sigma, \tau) \in \bigcup_{i\in[1,n]} \delta_i\}$, *where* $\hat{\rho}_\uparrow$ *and* $\hat{\rho}_\downarrow$ *are defined by* $\hat{\rho}_\uparrow(\sigma) = \sigma \in \mathrm{Dom}(\rho_\uparrow)\ ?\ \rho_\uparrow(\sigma) : \sigma$, $\hat{\rho}_\downarrow(\sigma) = \sigma \in \mathrm{Dom}(\rho_\downarrow)\ ?\ \rho_\downarrow(\sigma) : \sigma$;
- $S = \bigcup_{i\in[1,n]} S_i \setminus \mathrm{Dom}(\omega)$;
- $D_\uparrow \subseteq \bigcup_{i\in[1,n]} D_{\uparrow i}$, $D_\downarrow \subseteq \bigcup_{i\in[1,n]} D_{\downarrow i}$;
- $R_\uparrow = \bigcup_{i\in[1,n]} R_{\uparrow i} \setminus \mathrm{Dom}(\rho_\uparrow)$, $R_\downarrow = \bigcup_{i\in[1,n]} R_{\downarrow i} \setminus \mathrm{Dom}(\rho_\downarrow)$.

In this case, the P modules in $\mathcal{P}$ *are called* components *of* Ψ. *We omit here the straightforward but lengthy details of the required translations of the rulesets. Note that we can keep any of the previous external definitions, even those matched by external references (for further matches), thus the instantiations alone do not completely define the composition result.*

This modular approach provides encapsulation, information hiding and recursive composition, facilitating the design of P programs for complex algorithms. We now give a more elaborated example, where we use modules both to build a more complex system and to argue about its properties.

Example 2 (A composite P module for computing the GCD).
The left diagram of Figure 4 illustrates a logical flowchart for computing the standard Euclidean *Greatest Common Divisor* (GCD) algorithm, where, initially, x and y are the two positive integer inputs, and, on termination, the final value of y is the resulting GCD. For illustrative purposes, this design is recursively built, starting from four elementary blocks, Π_1, Π_2, Π_3 and Π_4. We first combine Π_1, Π_2 and Π_3 into a high-level block Π_5, and then combine Π_5 and Π_4 into a higher-level block Π_6. Our correctness proofs can start from the elementary blocks and then follow up this recursive composition.

The right diagram of Figure 4 illustrates a closely related recursive composition of four elementary P modules, Π_1, Π_2, Π_3 and Π_4, which solves the same problem (modulo a straightforward encoding).

Module $\Pi_1\langle \mathtt{def}_\downarrow\ \sigma_1, \mathtt{ref}_\downarrow\ Y_1, \mathtt{ref}_\downarrow\ N_1\rangle$ contains a single cell, σ_1, which also appears as an external $\mathtt{def}_\downarrow$ definition, and makes external $\mathtt{ref}_\downarrow$ references to two unspecified cells, Y_1 and N_1. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_1 receives x copies of a and y copies of b, then Π_1 ends by sending x copies of a and y copies of b; to Y_1, if $x \geq y$; or to N_1, otherwise.

1. $s_0\ ab \rightarrow_{\mathtt{max}}\ s_1\ cde$
2. $s_1\ a \rightarrow_{\mathtt{max}}\ s_2\ c$
3. $s_1\ b \rightarrow_{\mathtt{max}}\ s_3\ d$
4. $s_1\ e \rightarrow_{\mathtt{max}}\ s_2$
5. $s_1\ e \rightarrow_{\mathtt{max}}\ s_3$
6. $s_2\ c \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{Y_1}}$
7. $s_2\ d \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{Y_1}}$
8. $s_3\ c \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{N_1}}$
9. $s_3\ d \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{N_1}}$

Module $\Pi_2\langle \mathtt{def}_\downarrow\ \sigma_2, \mathtt{ref}_\downarrow\ Z_2\rangle$ contains a single cell σ_2, which also appears as an external $\mathtt{def}_\downarrow$ definition, and makes external $\mathtt{ref}_\downarrow$ references to one unspecified cell, Z_2.

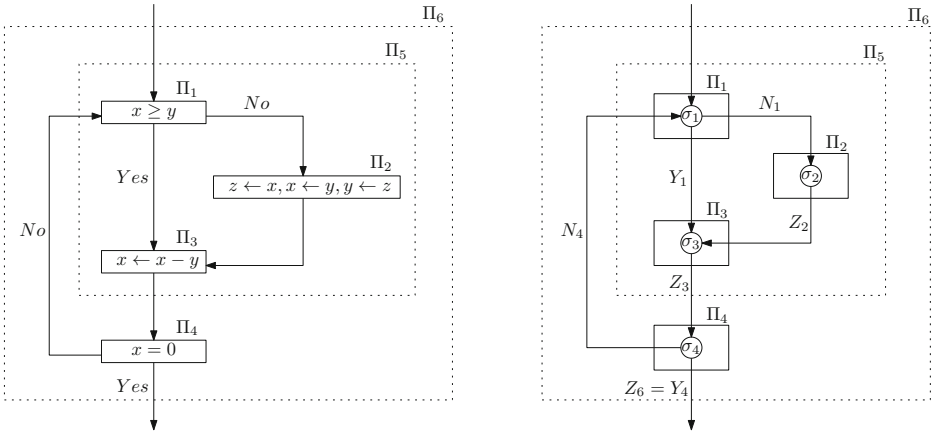


Fig. 4. Diagrams for computing GCD: left, a logical flowchart; right, a recursive composition of P modules

Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_2 receives x copies of a and y copies of b, then Π_2 ends by sending, to Z_2, y copies of a and x copies of b.

1. $s_0\ a \rightarrow_{\texttt{max}}\ s_0\ (b)_{\downarrow_{Z_2}}$
2. $s_0\ b \rightarrow_{\texttt{max}}\ s_0\ (a)_{\downarrow_{Z_2}}$

Module $\Pi_3\langle \texttt{def}_{\downarrow}\ \sigma_3, \texttt{ref}_{\downarrow}\ Z_3\rangle$ contains a single cell σ_3, which also appears as an external $\texttt{def}_{\downarrow}$ definition, and makes external $\texttt{ref}_{\downarrow}$ references to one unspecified cell, Z_3. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_3 receives x copies of a and y copies of b, then Π_3 ends by sending, to Z_3, x' copies of a and y' copies of b, where $x' = x - \min(x, y)$, $y' = \min(x, y)$.

1. $s_0\ ab \rightarrow_{\texttt{max}}\ s_0\ (b)_{\downarrow_{Z_3}}$
2. $s_0\ a \rightarrow_{\texttt{max}}\ s_0\ (a)_{\downarrow_{Z_3}}$

Module $\Pi_4\langle \texttt{def}_{\downarrow}\ \sigma_4, \texttt{ref}_{\downarrow}\ Y_4, \texttt{ref}_{\downarrow}\ N_4\rangle$ contains a single cell σ_4, which also appears as an external $\texttt{def}_{\downarrow}$ definition, and makes external $\texttt{ref}_{\downarrow}$ references to two unspecified cells, Y_4 and N_4. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_4 receives x copies of a and y copies of b, then, if $x = 0$, Π_4 ends by sending, to Y_4, y copies of c; or, if $x \neq 0$, Π_4 ends by sending, to N_4, x copies of a and y copies of b.

1. $s_0\ a \rightarrow_{\texttt{max}}\ s_1\ a$
2. $s_0\ b \rightarrow_{\texttt{max}}\ s_0\ (c)_{\downarrow_{Y_4}}$
3. $s_1\ a \rightarrow_{\texttt{max}}\ s_0\ (a)_{\downarrow_{N_4}}$
4. $s_1\ b \rightarrow_{\texttt{max}}\ s_0\ (b)_{\downarrow_{N_4}}$

We first combine Π_1, Π_2 and Π_3, using the following generic instantiations: $Y_1 = \sigma_3, N_1 = \sigma_2, Z_2 = \sigma_3$; this instantiates the connecting arcs (σ_1, σ_3), (σ_1, σ_2) and

(σ_2, σ_3). The result is a composite module with two generic parameters, an external definition and an external reference, $\Pi_5\langle \texttt{def}_\downarrow\ \sigma_1, \texttt{ref}_\downarrow\ Z_3\rangle$. Module Π_5's behaviour can be inferred from the behaviour of its constituent modules, Π_1, Π_2 and Π_3. If cell σ_1 receives x copies of a and y copies of b, then Π_5 ends by sending, to Z_3, x' copies of a and y' copies b, where $x' = \max(x, y) - \min(x, y)$, $y' = \min(x, y)$.

We further combine Π_5 and Π_4, using the following generic instantiations: $Z_3 = \sigma_4, N_4 = \sigma_1$; this instantiates the connecting arcs (σ_3, σ_4) and (σ_4, σ_1). The result is another composite module with two generic parameters, an external definition and an external reference, $\Pi_6\langle \texttt{def}_\downarrow\ \sigma_1, \texttt{ref}_\downarrow\ N_4\rangle$, which can also be renamed as $\Pi_6\langle \texttt{def}_\downarrow\ \sigma_1, \texttt{ref}_\downarrow\ Z_6\rangle$. Module Π_6's behaviour can be inferred from the behaviour of its constituent modules, Π_5 and Π_4. If cell σ_1 receives x copies of a and y copies of b, then Π_6 ends by sending, to Z_6, z copies of c, where $z = \gcd(x, y)$.

5 Revised Byzantine Agreement Solution

Our revised P solution for the Byzantine agreement problem is still based on Exponential Information Gathering (EIG) trees, for N processes connected in a complete graph, with "hardcoded" parameters L, the EIG tree height, and W, the default value (for missing or wrong messages). Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules: one "main" module, Ψ_h, which provides the main EIG functionality; plus one "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$. Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

Elementary modules are illustrated in Figure 5. Module Ψ_h has a single cell, ψ_h, which is also offered as an external definition, $\langle \texttt{def}_\downarrow\ \psi_h\rangle$. Module Γ_{hf} has three cells, γ_{hf}, γ'_{hf}, γ''_{hf}, offers one external definition $\langle \texttt{def}_\downarrow\ \gamma'_{hf}\rangle$, and uses two external references $\langle \texttt{ref}_\downarrow\ \psi_h, \texttt{ref}_\downarrow\ \gamma'_{fh}\rangle$.

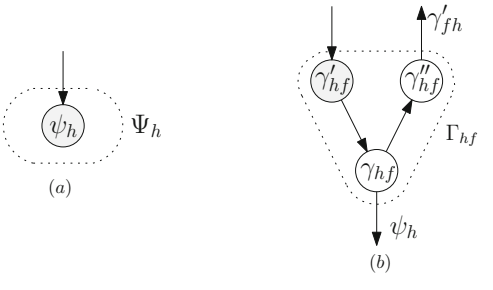


Fig. 5. Elementary P modules for Byzantine agreement: (a) main module Ψ_h, (b) communication module Γ_{hf}. The $\texttt{ref}_\downarrow$ references are indicated by labels on outgoing arcs and the $\texttt{def}_\downarrow$ definitions are indicated by shaded cells.

As mentioned, process module Π_h is composed from modules $\{\Psi_h\} \cup \{\Gamma_{hf} \mid f \in [1, N]\}$; this composition instantiates arcs $\{(\gamma_{hf}, \psi_h) \mid f \in [1, N]\}$. Module Π_h

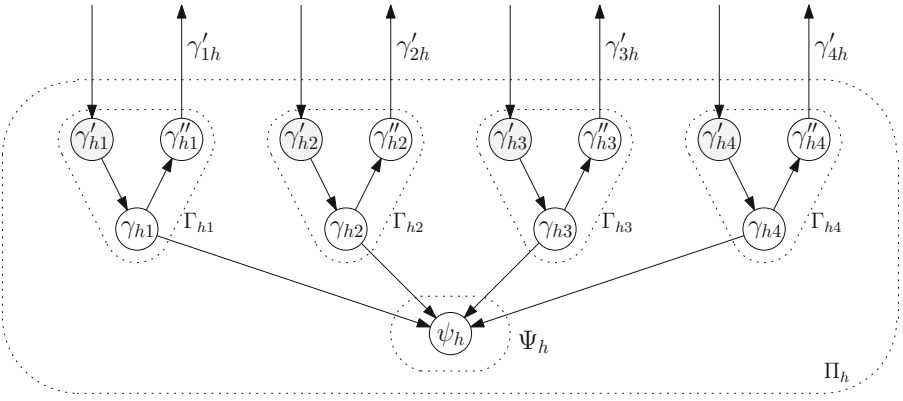


Fig. 6. The process module Π_h, for $N = 4$, $L = 2$. Its $\texttt{ref}_\downarrow$ references are indicated by labels on outgoing arcs (γ'_{1h}, γ'_{2h}, γ'_{3h}, γ'_{4h}) and its $\texttt{def}_\downarrow$ definitions are indicated by shaded cells (γ'_{h1}, γ'_{h2}, γ'_{h3}, γ'_{h4}).

offers N external definitions, $\langle \texttt{def}_\downarrow\ \gamma'_{hf} \mid f \in [1, N] \rangle$, and uses N external references, $\langle \texttt{ref}_\downarrow\ \gamma'_{fh} \mid f \in [1, N] \rangle$. Composite module Π_h, for $N = 4$, $L = 2$, is illustrated in Figure 6.

Any arbitrary module can play the role of a *faulty* module; however, to provide maximal adversity, it needs connection facilities similar to the expected facilities of non-faulty module. Therefore, without loss of generality, we model faulty processes by arbitrary modules Θ_h, $h \in [1, N]$, which offer N $\texttt{def}_\downarrow$ definitions and use N $\texttt{ref}_\downarrow$ references.

The final system is the composition of N modules from $\{\Pi_h \mid h \in [1, N]\} \cup \{\Theta_h \mid h \in [1, N]\}$, that instantiates arcs $\{(\gamma''_{hf}, \gamma'_{fh}) \mid h, f \in [1, N]\}$. The Byzantine agreement problem can be solved if at least $\lfloor 2(N-1)/3 \rfloor$ of these modules are non-faulty, i.e. from the Π_h family. Figure 7 shows a fragment with two modules, Π_2 and Π_3, of the four process modules of the final system, for the case $N = 4$, $L = 2$.

6 Rules and Correctness

The following objects are used by all non-faulty processes: $\Omega = \{x^v_\pi \mid v \in \{0, 1, ?, *\}, t \in [0, L-1], \pi \in P(N, t)\} \cup \{x^v_\pi \mid v \in \{0, 1\}, \pi \in P(N, L)\}$. Object x^v_π is viewed as an *encoding* of pair (π, v), which associates a permutation π, i.e. an EIG node, with a value v; object x^v_λ can be abbreviated as v. Encodings with binary digit values, x^v_π, $v \in \{0, 1\}$, are called *value objects* and represent EIG nodes with known α^h or β^h values. Where process number h is clearly inferred from the context, we will use α and β instead of α^h and β^h, respectively. Encodings with asterisks, x^*_π, are called *placeholder objects*, and represent EIG nodes with still undefined β^h values. Encodings with question marks, $x^?_\pi$, are called *template objects*, and are used to filter incoming messages which are not well-formed.

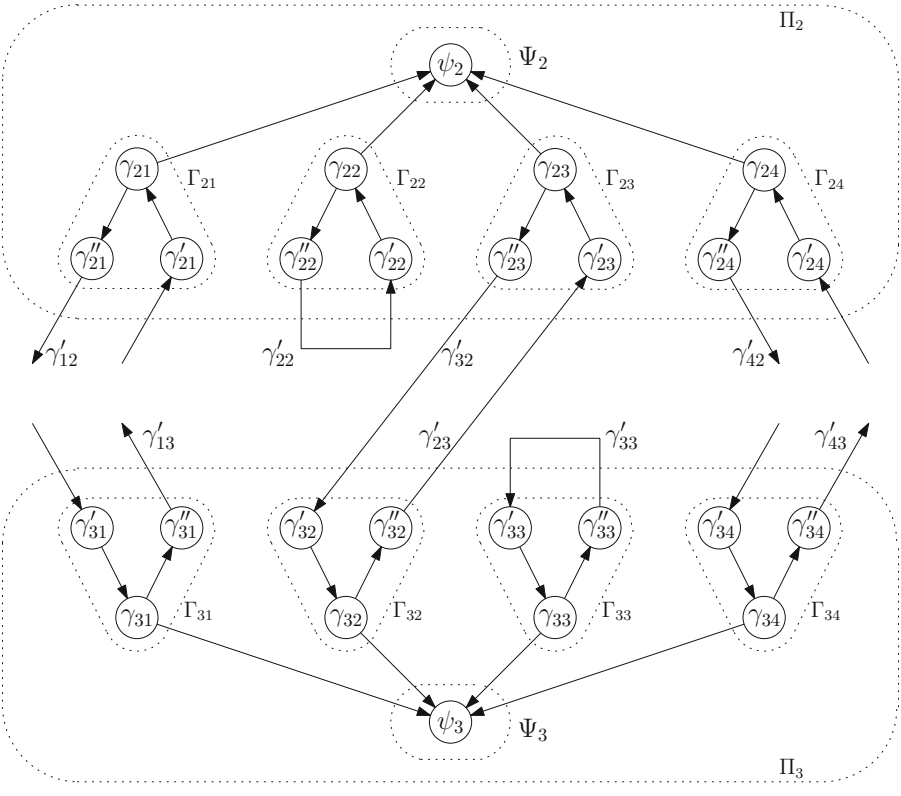


Fig. 7. Interconnection details between process modules Π_2 and Π_3, for $N = 4$, $L = 2$

Besides encodings in Ω, faulty process can send any other available *objects*. The set of all possible objects is denoted by a universal set $\mho$, i.e. $\mho = \Omega \cup \{$*all other objects which can be sent by a faulty process*$\}$.

The following sets of objects, which appear in several of the subsequent sections, are defined here:

$$R_t^{\alpha^h} = \{x_\pi^{\alpha^h(\pi)} \mid \pi \in P(N,t)\}, t \in [0,L]$$
$$R_t^{\beta^h} = \{x_\pi^{\beta^h(\pi)} \mid \pi \in P(N,t)\}, t \in [0,L]$$
$$R_t^{v} = \{x_\pi^{v} \mid \pi \in P(N,t)\}, t \in [0,L], v \in \{0,1,?,*\}$$
$$R_t^{0,1} = \{x_\pi^{v} \mid \pi \in P(N,t), v \in \{0,1\}\}, t \in [0,L]$$

The first three sets, $R_t^{\alpha^h}$, $R_t^{\beta^h}$, R_t^{v}, describe functional relations, on the underlying permutation π; the last one, $R_t^{0,1}$, is not. Where h can be unambiguously inferred from the context, the superscript h can be dropped from attribute names, α and β, i.e. in such cases, $R_t^{\alpha} = R_t^{\alpha^h}$, $R_t^{\beta} = R_t^{\beta^h}$.

Our technical report [7] provides specialized versions of these template rules, for the case $N = 4$ and $L = 2$.

6.1 Rule Sequence for Ψ_h's Cell ψ_h

According to the following rules, and as illustrated in the diagram of Figure 8, cell ψ_h progresses through states s_0 (start state), $s_1, \ldots, s_L, e_L, e_{L-1}, \ldots, e_0$ (final state).

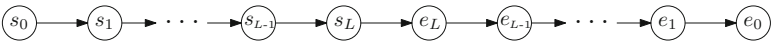


Fig. 8. State diagram for module Ψ_h, i.e. cell ψ_h

1. $s_t\ x^v_\pi \rightarrow_{\texttt{min}} s_{t+1}\ x^*_\pi\ (x^v_\pi x^?_\pi)_{\uparrow_{\texttt{repl}}}$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t)$
2. $s_L\ x^v_\pi \rightarrow_{\texttt{min}} e_L\ x^v_\pi$, for $v \in \{0,1\}, \pi \in P(N,L)$
3. $e_{t+1}\ x^0_{\pi\oplus k} x^1_{\pi\oplus l} \rightarrow_{\texttt{max}} e_t$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t), k,l \in [1,N] \setminus Im(\pi), k \neq l$
4. $e_{t+1}\ x^v_{\pi\oplus k} x^*_\pi \rightarrow_{\texttt{min}} e_t\ x^v_\pi$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t), k \in [1,N] \setminus Im(\pi)$
5. $e_{t+1}\ x^v_{\pi\oplus k} \rightarrow_{\texttt{max}} e_t$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t), k \in [1,N] \setminus Im(\pi)$
6. $e_{t+1}\ x^*_\pi \rightarrow_{\texttt{min}} e_t\ x^W_\pi$, for $t \in [0, L-1], \pi \in P(N,t)$

Initially, cell ψ_h is in state s_0 and contains a value object describing its initial choice, v^h. Cell ψ_h is a $\texttt{def}_\downarrow$ definition, thus, if properly connected, is able to receive and send objects from/to one or more parent cells, belonging to one or more "parent" modules. Cell ψ_h works in two phases, which roughly correspond to the two phases of the classical EIG-based algorithm: first, a *messaging phase*, implemented by rule groups 1 and 2, and, secondly, a bottom-up phase, implemented by rule groups 3, 4, 5 and 6.

The *external* behaviour of cell ψ_h's messaging phase, and therefore of module Ψ_h, is governed by the external contract described by the following paragraph.

Module Ψ_h's messaging phase takes $L+1$ rounds, indexed by $[0, L]$. The first round, 0, starts immediately, triggered by the presence of the initial value object. Each other round t, $t \in [1, L]$, starts after receiving, collectively from its parent modules, the set $R_t^{\alpha^h}$. Each round t, except the last, $t \in [0, L-1]$, ends by sending up, to each parent module, by replication, the set $R_t^{\alpha^h} \cup R_t^?$. Each round t, $t \in [0, L]$, is completed in exactly one P step. Module Ψ_h is idle between successive rounds.

Each $x_\pi^{\alpha(\pi)}$ sent up is accompanied by a corresponding template object, $x_\pi^?$, which is used, by cell ψ_h's parents, to build a filter, for next round value objects.

Messaging rounds here have different granularity and boundaries than in Section 3: in the classical EIG algorithm, a round starts by sending and continues by receiving messages; here, a round is triggered by receiving objects (except the first round, which is triggered by the initial choice) and continues by processing and sending objects (except the last round, which does only processing). This explains why, here, this messaging phase has $L+1$ rounds, but the messaging phase of Section 3 has L rounds.

The *internal* behaviour of cell ψ_h is very important, during both phases. At all steps, cell ψ_h's contents can be viewed as forming a *virtual EIG tree*, similar, but not identical, to the classical EIG tree of process h. The nodes of this virtual tree are represented by objects from the sets R_t^α, R_t^β, R_t^*, $t \in [0, L]$.

This virtual tree is gradually built, in increasing level order, during the messaging phase, and it gradually shrinks, in decreasing level order, during the bottom-up phase. Messaging round t, $t \in [0, L]$, is triggered by receiving value objects R_t^α. Just before receiving R_t^α, the virtual tree is given by $\cup_{l\in[0,t-1]} R_l^*$. Then, receiving R_t^α extends this virtual tree to $(\cup_{l\in[0,t-1]} R_l^*) \cup R_t^\alpha$. Next, if $t \in [0, L-1]$, messaging round t transforms the tree by "replacing" α value objects by placeholders, i.e. the virtual tree is now given by $(\cup_{l\in[0,t]} R_l^*)$ (which maintains the invariant); otherwise, if $t = L$, the virtual tree is not changed.

Using rule groups 1 and 2, an induction argument on round number t, $t \in [0, L]$, shows that cell ψ_h maintains its external contract for round t, gradually builds its virtual tree as mentioned, and transits from state s_t to state s_{t+1}, except for round $t = L$, when it transits from state s_L to state e_L.

As an example, consider that cell ψ_3 corresponds to process 3 in the scenario of Example 1. Figures 9 (a,b,c,d,e) illustrate the gradual completion of the virtual EIG tree, via the three messaging rounds of cell ψ_3.

These trees are represented with the help of the following sets; $R_0^* = \{x_\lambda^*\}$, $R_1^* = \{x_1^*, x_2^*, x_3^*, x_4^*\}$, $R_0^\alpha = \{x_\lambda^1\}$, $R_1^\alpha = \{x_1^0, x_2^0, x_3^1, x_4^1\}$, $R_2^\alpha = \{x_{12}^0, x_{13}^0, x_{14}^1, x_{21}^0, x_{23}^0, x_{24}^0, x_{31}^0, x_{32}^1, x_{34}^1, x_{41}^1, x_{42}^1, x_{43}^1\}$. $R_2^\beta = R_2^\alpha$, $R_1^\beta = \{x_1^0, x_2^0, x_3^1, x_4^1\}$, $R_0^\beta = \{x_\lambda^0\}$. For a comparison, see also Figure 1.

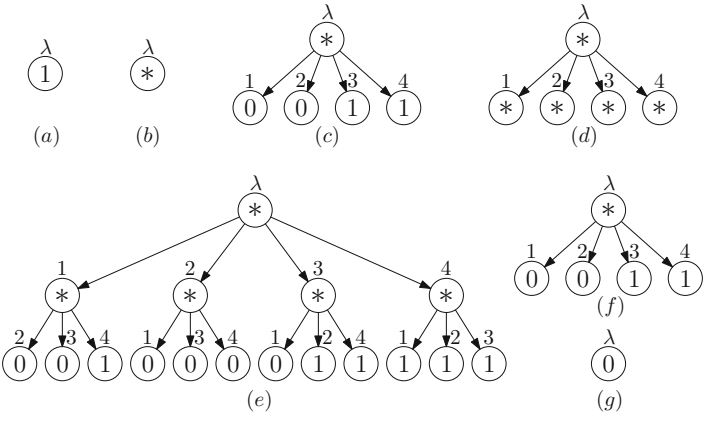


Fig. 9. The evolution and involution of cell ψ_3's virtual EIG tree. Message round 0: (a) ⇒ (b); Message round 1: (c) ⇒ (d); Message round 2: (d) ⇒ (e); Bottom-up step 1 : (e)⇒ (f); Bottom-up step 2 : (f) ⇒ (g).

There are $L + 1$ messaging rounds. Cell ψ_h completes each round in one P step and stays idle between rounds. Thus, the messaging phase will take $L(1 + \theta) + 1$ steps in total, where θ is the interval when cell ψ_h is idle, assumed fixed. (As we will see later, this interval is indeed fixed, $\theta = 4$, making a total of five P steps, required for sending objects from cell ψ_h to another cell ψ_f, $f \in [1, N]$, and vice-versa).

State e_L triggers the start of the *bottom-up evaluation* of attribute β, on the virtual EIG tree, in decreasing level order. Each level evaluation takes exactly one P step. Because $\beta(\pi) = \alpha(\pi)$, for $\pi \in P(N, L)$, the virtual tree at state e_L can be alternatively

viewed as $(\cup_{t\in[0,L-1]} R^*_t) \cup R^\beta_L$. An induction argument on $t = L, L-1, \ldots, 1, 0$, shows that, after $L-t$ steps since it has reached state e_L, cell ψ^h transits to state e_t and the virtual tree "shrinks" to $(\cup_{u\in[0,t-1]} R^*_u) \cup R^\alpha_t$.

Intuitively, at each transition from e_{t+1} to e_t, the β value objects for level $t+1$, R^β_{t+1}, are removed, and the placeholder objects for level t, R^*_t, are "replaced" by β value objects for level t, R^β_t.

Rule groups 3, 4, 5 and 6 run the required strict majority voting scheme, transiting from state e_{t+1} to state e_t. For each EIG sibling group at level $t+1$,

- Rule group 3 cancels pairs of β value objects at level $t+1$ with opposite binary values, until there either remains only objects with the same binary value, or no β value objects at all.
- Rule group 4 takes one of the remaining β value objects at level $t+1$, if there is a strict majority, and creates a corresponding β value object at level t.
- Rule group 5 removes all superfluous remaining β value objects at level $t+1$.
- Rule group 6 is activated in the tie case and creates a β value objects at level t with the default value W.

The last step of the bottom-up iteration evaluates R^β_λ, which contains a single β value object, $x^{\beta(\lambda)}_\lambda$, where $\beta(\lambda)$ is the final decision of process h. At the same time, cell ψ_h stops, because it reaches the final state e_0.

Figures 9 (e,f,g) continue the previous example, based on process 3 of the scenario of Example 1, and illustrate how the virtual tree "shrinks" after each step of the bottom-up evaluation. Note the tie breaker required for the last bottom-up step.

Including the root, the EIG tree has $L+1$ levels. Therefore, after receiving and processing the last round objects, which records the $\beta(\pi) = \alpha(\pi)$ values, for the leaves π, $\pi \in P(N, L)$, cell ψ_h needs L more P steps to reach the final state e_0 and evaluate the final decision value.

6.2 Rule Sequences for Γ_{hf}

Conceptually, module Γ_{hf} belongs to process h and stands as a local firewall, between its main module Ψ_h and a corresponding firewall module Γ_{fh}, belonging to untrusted remote process f.

Module Γ_{hf} contains three distinct cells, γ'_{hf}, γ''_{hf}, γ_{hf}, each having its own rule sequence and states. As indicated before, on one side (the "home" side), module Γ_{hf} is connected to the main module Ψ_h, and, on the other side (the "foreign" side), module Γ_{hf} expects to be connected to module Γ_{fh} (part of a "friend-or-foe" process f). Specifically, cell γ_{hf} is connected as parent of main cell ψ_h (which is given as an external reference), cell γ''_{hf} is connected as parent of foreign cell γ'_{fh} (which is given as an external reference), and cell γ'_{hf} (which is given as an external definition) is connected as child of foreign cell γ''_{fh}.

As will be shown in the next three subsections, cells γ'_{hf}, γ''_{hf}, γ_{hf}, work in lockstep, cycling continuously through a five P steps period, each period corresponding to a complete messaging round. As shown in Section 6.1, cell ψ_h completes its messaging related tasks in short one P step activity bursts, thus module Ψ does not have its

own time constraints for the messaging phase. Therefore, (a) the overall progress of module Ψ_h, during the messaging phase, is also determined by module Γ_{hf}, and (b) between successive messaging rounds, cell ψ_h stays idle for exactly four P steps, and the parameter θ, used in Section 6.1, is 4).

If counter s designates the global step number, $s = 1, 2, \ldots$, then the messaging round number is given by counter $t = (s-1)/5$, and counter $u \in [0, 4]$, defined by $u = s - 1 \pmod 5$, indicates the current substep inside the five steps period, which is also indicated by their current state index (e.g., cell γ'_{hf}'s states are indexed as c_u, $u \in [0, 4]$). Module Γ_{hf}'s cells and their external connections are expected to switch their responsibilities according to this counter u. Provided that both processes, h and f, are *non-faulty*, the expected messaging workflow of module Γ_h can be summarized as follows:

1. when $u = 0$, external cell ψ_h is expected to send up, to cell γ_{hf}, the object set $R_t^{\alpha^h} \cup R_t^?$;
2. when $u = 1$, cell γ_{hf} is expected to send down, to cell γ''_{hf}, the object set $R_t^{\alpha^h}$;
3. when $u = 2$, cell γ''_{hf} is expected to send down, to external cell γ'_{fh}, the object set $R_t^{\alpha^h}$, and, vice-versa, external cell γ''_{fh} is expected to send down, to cell γ'_{hf}, the object set $R_t^{\alpha^f}$;
4. when $u = 3$, cell γ'_{hf} is expected to send down, to cell γ_{hf}, the object set $R_t^{\alpha^f}$;
5. when $u = 4$, cell γ_{hf} is expected to send down, to cell ψ_h, the object set $R_t^{\alpha^f}$.

If these expectations are not met, i.e. if the foreign process f is faulty, module Γ_{hf} works as firewall, protecting its associated main module Ψ_h against bad, wrongly timed and missing messages. The message flow will not stop and, instead of bad or expected but missing objects, module Ψ_h will timely receive objects recreated with the default value W. A faulty process f might receive back some of the wrong messages it has itself sent to h, but this does not harm the algorithm.

Figure 10 illustrates a fragment of this workflow, by tracing the actual messages between cells ψ_2, ψ_3, the main cells associated to processes 2 and 3, respectively, in the Byzantine scenario of Example 1.

Module Γ_{hf} is hardwired for given level L. After L messaging phases, cell γ_{hf} stops working and enters its final state. The other two cells, γ'_{hf} and γ''_{hf} continue to loop over their five step cycles. This can be easily fixed, if this is not desired. However, this will involve a state-based countdown counter, because these two cells are on the frontline towards an untrusted process f, which potentially can alter their contents at any time.

The following three subsections detail the rules of module Γ_{hf}'s three cells and discuss their behaviour.

6.3 Rule Sequence for Γ_{hf}'s Cell γ'_{hf}

According to the following rules, and as illustrated in Figure 11 (a), cell γ'_{hf} cycles continuously through states c_0 (start state), c_1, c_2, c_3, c_4.

1. $c_0 \rightarrow_{\texttt{min}} c_1$
2. $c_1 \rightarrow_{\texttt{min}} c_2$

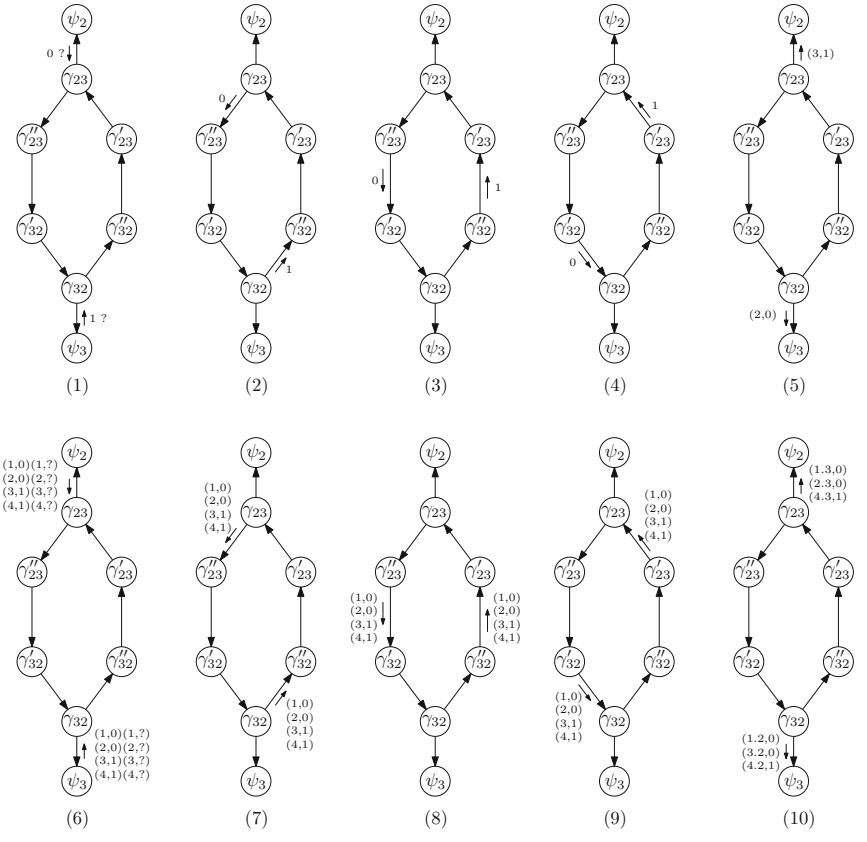


Fig. 10. Traces of the messaging phase between main cells ψ_2 and ψ_3, in the Byzantine scenario of Example 1

3. $c_2 \rightarrow_{\texttt{min}} c_3$
4. $c_2\ o \rightarrow_{\texttt{max}} c_3$, for $o \in \mho$
5. $c_3 \rightarrow_{\texttt{min}} c_4$
6. $c_3\ x_\pi^v \rightarrow_{\texttt{min}} c_4\ (x_\pi^v)_{\downarrow_{\gamma_{hf}}}$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t)$
7. $c_4 \rightarrow_{\texttt{min}} c_0$

Assume that s, t and u, are the correlated counters, defined in Section 6.2. When $u = 2$, cell γ'_{hf} is in state c_2, clears its contents by rule group 4 (practically useful, but not strictly needed) and expects $R_t^{\alpha f}$ from γ''_{fh}, if process f is non-faulty; however, a faulty process f may send, at any time, any objects in $\mho$. Next, when $u = 3$, cell γ'_{hf} is in state c_3 and, by rule group 6, forwards, to γ_{hf}, $R \cap (\cup_{l \in [0,L-1]} R_l^{0,1})$, a filtered subset of the actually received object set, R. This filter is "good-enough", but not complete, because it does not ensure that the forwarded objects form a functional relation on π or that π matches the current message round t. However, this mechanism protects cell γ_{hf} against wrongly timed objects or bad objects which could corrupt its internal consistency. As

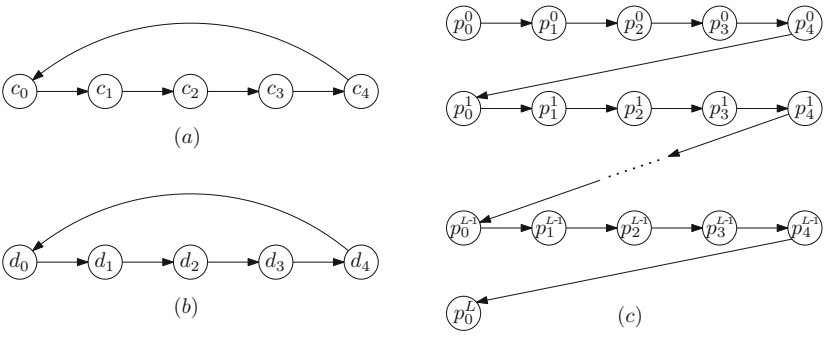


Fig. 11. State diagrams for module Γ_{hf}: (a) for cell γ'_{hf}, (b) for cell γ''_{hf}, (c) for cell γ_{hf}

we will see, cell γ_{hf} is able to solve the remaining formatting issues. For other values of u, cell γ'_{hf} keeps cycling, without doing any other significant work.

6.4 Rule Sequence for Γ_{hf}'s Cell γ''_{hf}

According to the following rules, and as illustrated in Figure 11 (b), cell γ''_{hf} cycles continuously through states d_0 (start state), d_1, d_2, d_3, d_4.

1. $d_0 \rightarrow_{\texttt{min}} d_1$
2. $d_1 \rightarrow_{\texttt{min}} d_2$
3. $d_1\ o \rightarrow_{\texttt{max}} d_2$, for $o \in \mho$
4. $d_2\ x_\pi^v \rightarrow_{\texttt{max}} d_3\ (x_\pi^v)_{\downarrow_{\gamma'_{fh}}}$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t)$
5. $d_3 \rightarrow_{\texttt{min}} d_4$
6. $d_4 \rightarrow_{\texttt{min}} d_0$

Assume that s, t and u, are the correlated counters, defined in Section 6.2. When $u = 1$, cell γ''_{hf} is in state d_1, clears its contents by rule group 3 (practically useful, but not strictly needed) and expects $R_t^{\alpha^h}$ from γ_{hf}. Next, when $u = 2$, cell γ''_{hf} is in state d_2 and, by rule group 4, forwards down, to γ'_{fh}, the previously received objects, i.e. $R_t^{\alpha^h}$. For other values of u, cell γ''_{hf} keeps cycling, without doing any other significant work.

6.5 Rule Sequence for Γ_{hf}'s Cell γ_{hf}

According to the following rules, and as illustrated in Figure 11 (c), cell γ_{hf} progresses through states p_1^0 (start state), p_2^0, p_3^0, p_4^0, p_0^1, p_1^1, p_2^1, p_3^1, p_4^1, ..., p_0^L (final state).

1. $p_0^t \rightarrow_{\texttt{min}} p_1^t$, for $t \in [0, L-1]$
2. $p_1^t \rightarrow_{\texttt{min}} p_2^t$, for $t \in [0, L-1]$
3. $p_1^t\ x_\pi^v \rightarrow_{\texttt{min}} p_2^t\ (x_\pi^v)_{\downarrow_{\gamma''_{hf}}}$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t)$
4. $p_2^t \rightarrow_{\texttt{min}} p_3^t$, for $t \in [0, L-1]$
5. $p_3^t \rightarrow_{\texttt{min}} p_4^t$, for $t \in [0, L-1]$

6. $p_4^t \rightarrow_{\texttt{min}} p_0^{t+1}$, for $t \in [0, L-1]$
7. $p_4^t\ x_\pi^? \rightarrow_{\texttt{min}} p_0^{t+1}\ (x^0_{\pi\oplus f})_{\downarrow_{\psi_h}}$, for $t \in [0, L-1]$, $\pi \in P(N,t)$, $f \notin Im(\pi)$
8. $p_4^t\ x_\pi^? \rightarrow_{\texttt{min}} p_0^{t+1}\ (x^1_{\pi\oplus f})_{\downarrow_{\psi_h}}$, for $t \in [0, L-1]$, $\pi \in P(N,t)$, $f \notin Im(\pi)$
9. $p_4^t\ x_\pi^? \rightarrow_{\texttt{min}} p_0^{t+1}\ (x^W_{\pi\oplus f})_{\downarrow_{\psi_h}}$, for $t \in [0, L-1]$, $\pi \in P(N,t)$, $f \notin Im(\pi)$
10. $p_4^t\ o \rightarrow_{\texttt{max}} p_0^{t+1}$, for $t \in [0, L-1]$, $o \in \Omega$

Assume that s, t and u, are the correlated counters, defined in Section 6.2. When $u = 0$, cell γ_{hf} is in state p_0^t, is empty, and expects $R_t^{\alpha^h} \cup R_t^?$ from ψ_h. Next, when $u = 1$, cell γ_{hf} is in state p_1^t and, by rule group 3, forwards down, to γ''_{hf}, the value objects destined to process f, $R_t^{\alpha^h}$ but keeps the template objects, $R_t^?$. Next, when $u = 2$, cell γ_{hf} is in state p_2^t and keeps cycling, by rule group 4, without doing any other significant work. Next, when $u = 3$, cell γ_{hf} is in state p_3^t and expects, from cell γ'_{hf}, a set of good-enough value objects, ideally $R_t^{\alpha^f}$, but could be any subset of $(\cup_{l\in[0,L-1]} R_l^{0,1})$.

Case $u = 4$ describes a critical step, where cell γ_{hf} uses the process number f to tag the objects which are forwarded to ψ_h. Consider the partial function $\mathcal{S}_f : \Omega \rightarrow \Omega$, defined by $\mathcal{S}_f(x_\pi^v) = x^v_{\pi\oplus f}$, for $\pi \in (\cup_{l\in[0,L-1]} P(N,l))$, $v \in \{0,1\}$. Cell γ_{hf} is in state p_4^t and forwards down, to cell ψ_h, either (a) the set $\mathcal{S}_f(R_t^{\alpha^f})$, if process f is non-faulty; or, (b) a reconstructed version, where unavailable objects are replaced by value objects, reconstructed with the default value W.

The correct format is checked by matching received value objects against template objects $R_t^?$. Rule group 7 applies when a 0 valued object can be matched against a template. Rule group 8 applies when a 1 valued object can be matched against a template. Rule group 9 applies when no value object matched the template and recreates a missing value object based on the default value W. After this, cell γ_{hf} clears all its contents, by rule group 10, preparing itself for the next cycle.

6.6 Module Π_h

Collecting the above results, we can now complete the last item required by the contract between the main module, Ψ_h, with its firewall modules, Γ_{hf}, $f \in [1, N]$. Assume again that s, t and u, are the correlated counters, defined in Section 6.2 and $t \in [0, L-1]$. As shown in Section 6.5, when $u = 4$, for each $h, f \in [1, N]$, cell Γ_{hf} sends down, to cell ψ_h, either (a) the set $\mathcal{S}_f(R_t^{\alpha^f})$, if process f is non-faulty; or, (b) a reconstructed version, where unavailable objects are replaced by value objects, reconstructed with the default value W.

Therefore, cell ψ_h receives, collectively from its parent modules, the set $S = \cup_{f\in[1,N]} \mathcal{S}_f(R_t^{\alpha^f})$. A straightforward argument shows that sets $\mathcal{S}_f(R_t^{\alpha^f})$, $f \in [1, N]$, are disjoint, and their union S is $S = R_{t+1}^{\alpha^h}$. This triggers the next messaging round $t + 1$, with the required new set of value objects, which completes the argument.

6.7 Complexity

As indicated by the next theorem, this new version of the Byzantine algorithm improves the runtime of the previous version [8], from $9L + 6$ to $6L + 1$, where, typically, $L = \lceil N/3 \rceil$.

Theorem 1. *This revised EIG-based Byzantine algorithm takes* $6L + 1$ *steps, where* L *is the number of messaging rounds.*

Proof. As seen above, it takes $5L$ P steps from start until the last message is received by the main cells, ψ_h, $h \in [1, N]$, One additional P step is required to transit from ψ_h's state s_L to state e_L, i.e. to transit from from the messaging phase to the bottom-up phase. Finally, cell ψ_h needs L more P steps for its bottom-up phase, to evaluate its final decision value and reach the final state. Thus, the revised Byzantine algorithm takes a total of $5L + 1 + L = 6L + 1$ steps.

The new version reduces the total number of cells required, from super-exponential, $O(N!)$, to a small polynomial, $O(N^2)$. However, some other static complexity measures are still very large. The new version does not change the message complexity of the previous version, which is mostly determined by the EIG algorithm itself. Table 1 summarizes these complexity measures.

Table 1. Summary of complexity measures (where, typically, $L = \lceil N/3 \rceil$)

Complexity measure	Previous version	Current version
Number of steps	$9L + 6$	$6L + 1$
Number of cells per Π module	$2N + 1 + O(N!)$	$3N + 1$
Number of objects	$O(N!)$	$O(N!)$
Maximum number of states per elementary module	$O(L)$	$O(L)$
Maximum number of rules per elementary module	$O(N!)$	$O(N!)$
Number of messages exchanged between Π modules	N^2L	N^2L

7 Conclusions and Open Problems

In this paper, we proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems, and used it to provide a faster P solution for the Byzantine agreement algorithm, based on Exponential Information Gathering (EIG) trees.

Our modular framework offers three types of generic parameters: generic objects, external definitions and external references and supports encapsulation, information hiding and modular composition.

Our revised P solution uses only duplex channels, fewer cells and fewer rules, while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds. We proved that modules, i.e. cell clusters, can solve the classical Byzantine agreement problem. Our design uses $3N + 1$ cells for each module, with one "main" cell and $3N$ ancillary cells, which enclose the main cell inside a "firewall". Can we solve the Byzantine agreement directly between individual cells, without help from any additional firewall?

In our case, firewall cells have a complex role. They protect the main cell against badly formatted, wrongly timed and missing messages. If they reach the main cell, wrongly timed bad messages have the potential to corrupt the internal structures, required by the internal cell logic. Additionally, our firewall cells tag incoming messages with unforgeable origin marks (a feature that current passive channels do not offer). This is a critical feature of the EIG-based algorithm itself (not of the cell implementing it). If the originator is not guaranteed, this algorithm will fail.

We believe that some of these firewall tasks can be retrofitted into the main cell itself, but not all required critical features. Thus, it seems that it is not possible to achieve a Byzantine agreement between individual cells, if we rely on the classical EIG-based algorithm.

However, there are many other algorithms for the Byzantine agreement, thus, our question is more general. Is there any other algorithm able to solve the Byzantine agreement at the cell level, still using passive channels, as in the current framework? We conjecture that the answer is negative. If this is indeed the case, what is the minimal size of one firewall component, one, two, three?

One can make a parallel to the history of the Internet. "The Internet protocols were originally designed for openness and flexibility, not for security. The ARPA researchers needed to share information easily, so everyone needed to be an unrestricted insider on the network" [9]. It seems that this was also the case in the early development of P systems. Are there simple ways to enhance our passive channels to provide more safety? Besides distributed computing, would this be useful in other modelling scenarios? A related problem, are there real-life biological scenarios, which need sophisticated fault-tolerant mechanisms, similar to the Byzantine agreement algorithms used in distributed computing, and, if yes, how do these really work?

Besides the above conjecture, our investigation leaves open a number of other interesting and challenging problems. Can we extend our P system solution to cover $2F + 1$ connected graphs, but not necessarily complete? Can we design P system solutions for other Byzantine agreement algorithms, not EIG-based, for example using reliable broadcasts? Will other solutions work "better", e.g., faster or with smaller communication overhead? Is it possible to solve the Byzantine agreement problem with a fixed number of P rules? If not, which is likely the case, can we solve this problem by proposing simple "natural" extensions to current rule system? Finally, can we provide solutions for some types of asynchronous P systems and what additional constraints will be needed in this case?

References

1. Abd-El-Malek, M., Ganger, G.R., Goodson, G.R., Reiter, M.K., Wylie, J.J.: Fault-scalable Byzantine fault-tolerant services. In: Herbert, A., Birman, K.P. (eds.) SOSP, pp. 59–74. ACM, New York (2005)
2. Ben-Or, M., Hassidim, A.: Fast quantum Byzantine agreement. In: Gabow, H.N., Fagin, R. (eds.) STOC, pp. 481–485. ACM, New York (2005)
3. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. J. Cryptology 18(3), 219–246 (2005)

4. Castro, M., Liskov, B.: Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4), 398–461 (2002)
5. Ciobanu, G.: Distributed algorithms over communicating membrane systems. Biosystems 70(2), 123–133 (2003)
6. Ciobanu, G., Desai, R., Kumar, A.: Membrane systems and distributed computing. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 187–202. Springer, Heidelberg (2003)
7. Dinneen, M.J., Kim, Y.-B., Nicolescu, R.: A faster P solution for the Byzantine agreement problem. Report CDMTCS-388, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland (July 2010)
8. Dinneen, M.J., Kim, Y.-B., Nicolescu, R.: P systems and the Byzantine agreement. The Journal of Logic and Algebraic Programming (in Press, 2010) Corrected Proof)
9. Froehlich, F.E., Kent, A.: Encyclopedia of Telecommunications, vol. 15. CRC Press, Boca Raton (1997)
10. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982)
11. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco (1996)
12. Martin, J.-P., Alvisi, L.: Fast Byzantine consensus. IEEE Trans. Dependable Sec. Comput. 3(3), 202–215 (2006)
13. Nicolescu, R., Dinneen, M.J., Kim, Y.-B.: Towards structured modelling with hyperdag P systems. International Journal of Computers, Communications and Control 2, 209–222 (2010)
14. Păun, G.: Membrane Computing: An Introduction. Springer, New York (2002)
15. Păun, G.: Introduction to membrane computing. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.) Applications of Membrane Computing. Natural Computing Series, pp. 1–42. Springer, Heidelberg (2006)
16. Păun, G., Pérez-Jiménez, M.J.: Solving problems in a distributed way in membrane computing: dP systems. International Journal of Computers, Communications and Control 5(2), 238–252 (2010)
17. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM 27(2), 228–234 (1980)
18. Romero-Campero, F.J., Twycross, J., Cámara, M., Bennett, M., Gheorghe, M., Krasnogor, N.: Modular assembly of cell systems biology models using P systems. Int. J. Found. Comput. Sci. 20(3), 427–442 (2009)
19. Serbanuta, T., Stefanescu, G., Rosu, G.: Defining and executing P systems with structured data in K. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 374–393. Springer, Heidelberg (2009)

Computationally Complete Spiking Neural P Systems without Delay: Two Types of Neurons Are Enough

Rudolf Freund[1] and Marian Kogler[1,2]

[1] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
{rudi,marian}@emcc.at
[2] Institute of Computer Science, Martin Luther University Halle-Wittenberg
Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany
kogler@informatik.uni-halle.de

Abstract. In this paper, we consider spiking neural P systems without delay with specific restrictions on the types of neurons. Two neurons are considered to be of the same type if the rules, the number of spikes in the initial configuration and the number of outgoing synapses are identical. We show that computational completeness can be achieved in both the generating and the accepting case with only two types of neurons, where the number of neurons with unbounded rules is constant even minimal.

1 Introduction

Spiking neural P systems, introduced by Ionescu et al. in [3], are a special class of P systems (see [7], [8], [12]) inspired by the working of the (human) brain. In spiking neural P systems, the cells represent neurons, which are connected by synapses. Spiking neural P systems have only one symbol, the *spike*, multiple copies of which can be sent ("fired") from neurons via synapses to other neurons or simply "forgotten" (i.e., removed). The exchange of spikes is regulated by firing and forgetting rules in the neurons.

The problem which "ingredients" are needed to achieve computational completeness or universality with spiking neural P systems has been a challenging question since the introduction of this kind of systems. Several answers have been given, for instance showing that many additional conditions such as delays can be left off [2] or that a limited number of rules per neuron suffices [1], [5], or giving universal P systems with a small number of neurons [6], [11].

Recently, Zeng et al. [10] have considered homogeneous spiking neural P systems, i.e. P systems where the rules in every neuron are identical. However, to achieve universality in this model, some extra conditions such as delays are required.

In our paper, we consider spiking neural P systems with only two types of neurons, without using delays. We consider two neurons to be of the same type if the rules, the number of spikes in the initial configuration, and the number

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 198–207, 2010.

of outgoing edges (synapses) are identical. We show that we are able to achieve computational completeness both in the accepting and in the generating case. Moreover, we are able to give a constant (even minimal) bound on the required number of neurons with unbounded rules.

The rest of the paper is organized as follows: in the next section, we consider some basic formalisms from formal language theory and give the definition of spiking neural P systems. In Section 3 we present our results regarding computational completeness. Finally, we conclude the paper with a short summary of the results obtained in this paper and some interesting open questions for future research.

2 Definitions

The reader is assumed to be familiar with basic notions of formal language theory (for instance, see [9]).

In the following, by NRE we denote the family of recursively enumerable sets of natural numbers. For a regular expression E, the corresponding regular language is denoted by $L(E)$.

A nondeterministic register machine is a construct

$$M = (n, B, p_0, p_h, I) \text{ where}$$

1. n, $n \geq 1$, is the number of registers,
2. B is the set of instruction labels,
3. p_0 is the start label,
4. p_h is the halting label (only used for the `HALT` instruction) and
5. I is a set of (labeled) instructions being of one of the following forms:
   - $p_i : (\texttt{ADD}(r), p_j, p_k)$ increments the value in register r and continues with one of the instructions labeled by p_j and p_k, chosen in a nondeterministic way,
   - $p_i : (\texttt{SUB}(r), p_j, p_k)$ tries to decrement the value in register r; if the register was non-empty before the instruction, the computation continues with the instruction labeled with p_j, if not, it continues with the instruction p_k;
   - $p_h : \texttt{HALT}$ halts the machine.

As the only nondeterminism occurs in the `ADD`-instructions, we can easily construct deterministic register machines by imposing the condition $p_j = p_k$. In this case, we write $p_i : (\texttt{ADD}(r), p_j)$. We will be using nondeterministic register machines as generators and deterministic register machines as acceptors.

A deterministic register machine accepts a (vector of) natural numbers by starting with the number(s) as input in the first register(s), with all other registers being empty. Starting from the instruction labeled with p_0, the instructions are applied and the contents of the registers changed; if and when the machine reaches p_h and therefore halts, the vector or number is accepted. Register machines can accept all recursively enumerable sets of (vectors of) natural numbers (with k components) with $k + 2$ registers (for instance, see [4]).

In the generating case, the (now nondeterministic) register machine starts with empty registers at the initial instruction p_0. When the machine halts, the contents of the first k registers form the result. Every recursively enumerable set of (vectors of) natural numbers can be generated with only $k+2$ registers, where the first k registers are never decremented [4].

2.1 Spiking Neural P Systems

A spiking neural P system (without delays) is a construct

$$\Pi = (O, \rho_1, ..., \rho_n, syn, in, out) \text{ where}$$

1. $O = \{a\}$ is the (unary) set of objects (the object a is called *spike*),
2. $\rho_1, ..., \rho_n$ are the neurons, where $\rho_i = (d_i, R_i)$ for $1 \leq i \leq n$, with d_i being the initial configuration of the neuron i and R_i being the set of rules, where the rules have one of the following forms:
   - $E/a^i \rightarrow a^j$, where E is a regular expression over O and $i, j \geq 1$ (*firing rules*) or
   - $a^i \rightarrow \lambda$, where $i \geq 1$ (*forgetting rules*).

   There must not be any rule $a^i \rightarrow \lambda$ such that $a^i \in L(E)$ for some E of a firing rule.
3. $syn \subseteq \{1, ..., n\} \times \{1, ..., n\}$ are the synapses, where $(i, j) \in syn$ indicates a synapse from i to j,
4. in is the input neuron (with the only function to spike once in generating spiking neural P systems in order to start a computation), and
5. out is the output neuron (no function in accepting spiking neural P systems).

A computation of a spiking neural P system starts from the initial configuration $(d_0, d_1, ..., d_n)$ (d_i represents the number of spikes in neuron i, $1 \leq i \leq n$, and d_0 the number of spikes in the environment) and then proceeds by making computation steps until the system halts, i.e., when in no neuron a rule can be applied any more. With the system being in the configuration $(c_0, c_1, ..., c_n)$, where c_i represents the number of spikes in neuron i, $1 \leq i \leq n$, and c_0 the number of spikes to be sent from the environment to the input neuron, a computation step is carried out as follows: in every neuron, one rule – if possible – is chosen in a nondeterministic way and applied. A firing rule $E/a^i \rightarrow a^j \in R_k$ can be applied in neuron k if, for $c_k = a^m$, $c_k \in L(E)$ and $m \geq i$, removing i spikes from k and adding j spikes in every neuron l where $(k, l) \in syn$. A forgetting rule $a^i \rightarrow \lambda$ can be applied in neuron k if and only if $c_k = a^i$; such a rule removes all spikes from the neuron.

Per step and neuron, only one rule may be applied. This is in contrast to other variants of P systems, which usually utilize the maximally parallel mode. As spiking and forgetting rules are mutually exclusive, the only way a nondeterministic computation step may happen is by the existence of two rules where the languages generated by their regular expressions have a non-empty intersection and the number of fired spikes is not equal.

A spiking neural P system inputs and outputs numbers via a spike train. A spike train starts with a spike given in step t_1 and ends with a spike given in step t_2. The number is specified by $t_2 - t_1 - 1$, i.e., the number of steps that elapse between the two spikes. It accepts an input by a series of configurations, starting from the initial configuration and ending in a halting configuration.

Rules of the form $E/a^i \to a^j$ where $L(E)$ is finite (infinite) are called *bounded (unbounded) rules*.

Two neurons ρ_i and ρ_j are of the same type if and only if $R_i = R_j$, $d_i = d_j$ and $|\{(i,k) \in sym \mid k \in \{1,...,n\}\}| = |\{(j,k) \in sym \mid k \in \{1,...,n\}\}|$.

By $NSN_{acc}P_*(types_k)$ ($NSN_{gen}P_*(types_k)$) we denote the family of sets of natural number accepted (generated) by spiking neural P systems needing neurons of at most k different types.

3 Results

Theorem 1. $NSN_{acc}P_*(types_2) = NRE$.

Proof. We prove computational completeness by simulating deterministic register machines with three registers, where the first one is the input register. We use two types of neurons, where neurons of type 1 are responsible for most of the computations, and neurons of type 2 are equivalent to registers (if the register r contains the value i, the neuron r contains $2i$ spikes). Therefore, only three neurons of type 2 are required, whereas we have an unbounded number of neurons of type 1. Both types have exactly two outgoing synapses and λ (i.e. no spike) in the initial configuration. The two types of neurons and their ingredients are shown in Figure 1.

Fig. 1. The two types of neurons used in the proof of Theorem 1

In the following, we will depict cells of type 1 by squares and cells of type 2 by dashed squares without giving the rules or the initial configuration again.

As all neurons have an out-degree of two and the generation of additional spikes is only possible with extra steps, it is sometimes necessary to utilize dummy neurons. A dummy neuron structure (see Figure 2) takes in one spike and forgets it. A circle with the letter "D" will be used in the following to denote such a structure.

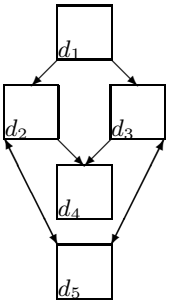


Fig. 2. A dummy neuron structure

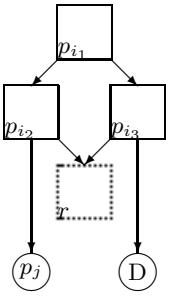


Fig. 3. A neuron structure simulating an ADD-instruction

When a spike enters neuron d_1, it triggers the rule $a/a \to a$, which causes the neuron to spike into the neurons d_2 and d_3, which both fire into the neurons d_4 and d_5, thus giving two spikes in both of these neurons, which are forgotten by the rule $a^2 \to \lambda$. As the neurons d_4 and d_5 never spike, the synapses from the neurons d_4 and d_5 back to the neurons d_2 and d_3 have no effect.

The simulation of an ADD instruction $p_i : (\texttt{ADD}(r), p_j)$ is accomplished by the neuron structure as shown in Figure 3: When a spike enters neuron p_{i_1}, the simulation starts. The neuron fires and sends a spike into p_{i_2} and p_{i_3}, which fire two spikes into r (thereby increasing the value in register r by 1), one spike into the neuron structure for p_j (to start the simulation of the instruction p_j), and one spike into a dummy structure. Therefore, the simulation of an ADD-instruction requires three neurons (not including the dummy structure) and two steps.

For SUB-instructions, we do not consider the instructions individually; rather, we construct a structure responsible for all SUB(r)-instructions. In the following, we will denote the SUB-instructions for some register r by $p_i : (SUB(r), p_{i_s}, p_{i_f})$ for $1 \leq i \leq n$, where the p_i are the instructions, the p_{i_s} are the follow-up instructions in case of success (i.e., when the register is non-empty) and the p_{i_f} the follow-up instructions in case of failure (i.e., when the register is empty). Figure 4 illustrates the neuron structure needed for the simulation of SUB(r)-instructions; p_i denotes the instruction currently being executed and n the number of SUB(r)-instructions. In this diagram, only one possible instruction is described in detail.

Dummy structures – used where there is only one non-dummy outgoing edge – have been left off. Moreover, the spike from register r has to be forwarded to all possible p'_{j_1} (for this, $\frac{\lceil\sqrt{n}\rceil(\lceil\sqrt{n}\rceil-1)}{2}$ neurons are needed) and the spike for the instruction needs to be delayed until this spike arrives (using $2(\lceil\sqrt{n}\rceil-1)$ cells).

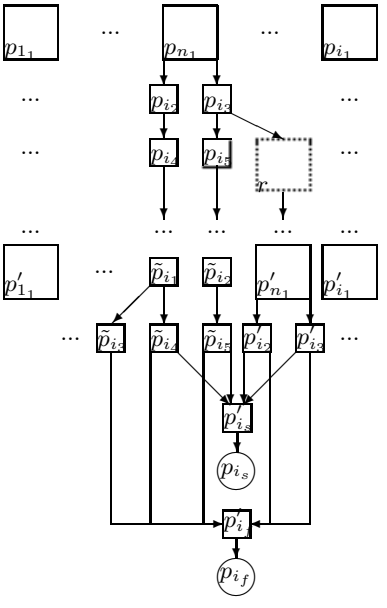


Fig. 4. A simplified diagram of a SUB-instruction neuron structure

When the simulation of the instruction starts, one spike is sent into the neuron representing register r. If the register is empty (i.e., the neuron contains no spike), the spike is forgotten by the rule $a \to \lambda$, if not, the rule $a^3(a^2)^*/a \to a$ removes two spikes (thus decrementing the value in the register) and fires one spike. However, as the register does not know which instruction caused the subtraction, this spike is forwarded to all possible instructions. In parallel, one spike is fired by the instruction neuron, to be duplicated and forwarded to the neuron of the instruction. To make sure that the spikes from the register and the spikes from the instruction reach the register at the same time, the two spikes from the instruction neuron (after duplication) are delayed by additional neurons of type 1. Finally, the neurons p'_{i_f} and p'_{i_s} act as "gatekeepers" of the respective instructions.

The neuron p'_{i_s} corresponding to the current instruction receives two spikes. If the subtraction succeeded (i.e., if the register was non-empty), every neuron p'_{i_s} receives two spikes. If only two spikes are present (i.e., if the neuron belongs to a different instruction or the operation was not successful), they are forgotten by the rule $a^2 \to \lambda$; otherwise, four spikes are in the neuron, which causes it to fire by the rule $a^4/a^4 \to a$ and therefore to start the simulation of the next instruction p_{i_s}. Moreover, the neuron p'_{i_f} receives three spikes from the current

instruction, which causes it to fire if the subtraction did not succeed and therefore to start the simulation of the next instruction p_{i_f}; if the subtraction succeeded, two additional spikes are sent from the register, triggering the forgetting rule $a^5 \to \lambda$. If only these two spikes representing a successful subtraction are sent in, the forgetting rule $a^2 \to \lambda$ can be applied and removes them.

The simulation of a SUB-instruction takes $5 + (\lceil\sqrt{n}\rceil - 1)$ steps and $15 + (\lceil\sqrt{n}\rceil - 1)$ neurons (not counting dummy structures). Additionally, $\frac{\lceil\sqrt{n}\rceil(\lceil\sqrt{n}\rceil-1)}{2}$ neurons for every register are needed, where n is the number of SUB-instructions that affect this register.

The HALT-instruction is represented by a dummy structure, which consumes the spike, thus ending the computation of the P system.

Finally, we consider the input to be given by two spikes: one in the first step and one in the $n+2$th step, where n is the input number. The neuron structure as depicted in Figure 5 puts $2n$ spikes into the input register r_i. When the first spike enters the system, it initializes the two loops and puts one spike into the register (which is forgotten because of the rule $a \to \lambda$). Per step, the first loop (neurons i_3 and i_4) puts two spikes into the register, thereby increasing the count by 1. The second loop (neurons i_{11} and i_{12}) puts two spikes into an intermediate neuron i_{13} (where they are forgotten immediately).

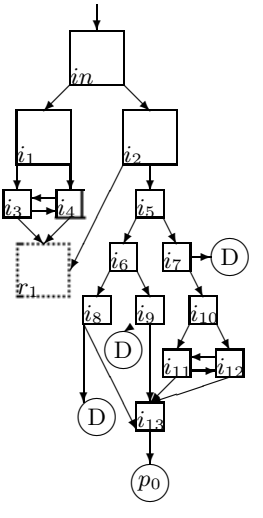


Fig. 5. The neuron structure initializing the computation

After the second spike has entered, the loops stop (as the forgetting rules $a^2 \to \lambda$ can be applied), another spike is fired into the register (as the input is the number of steps between the two spikes, but the loop runs one more time, it is necessary to decrement the register once) and four spikes enter i_{13}, thus causing it to fire a spike into the structure simulating the first instruction p_0.

For any number as input, the spiking neural P system halts if and only if the corresponding register machine halts. This observation concludes the proof. □

Theorem 2. $NSN_{gen}P_*(types_2) = NRE$.

Proof. Our construction is similar to the proof given for Theorem 1. However, we need to simulate nondeterministic register machines here, so we need to modify the simulation of ADD-instructions. To be able to simulate nondeterminism, we introduce two additional rules to the neuron type 1 (see Figure 6), which allows for the nondeterministic simulation of an instruction $p_i : (\texttt{ADD}(r), p_j, p_k)$ (see Figure 7).

Fig. 6. The two types of neurons used in the proof of Theorem 2

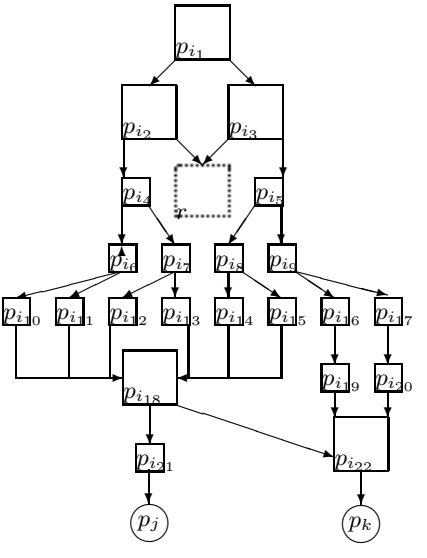


Fig. 7. A neuron structure simulating an ADD-instruction (without dummy structures)

After the instruction has been executed, the neuron $p_{i_{18}}$ contains six spikes, which nondeterministically triggers one of the rules $a^6/a^6 \to a$ and $a^6/a^6 \to a^2$. If the first rule is executed, the instruction p_j immediately follows; otherwise, the instruction p_k is executed. (This construction is vaguely similar to the one

used for SUB-instructions; however, in this case, both input spikes come from the same source.) This simulation requires 22 neurons (not counting the dummy structures) per ADD-instruction and is executed in seven steps.

The simulation of SUB-instructions is exactly as in the proof given in the proof of the previous theorem. However, we do not need to consider SUB-instructions for the output register 1, as we can assume that this first register is never decremented.

To start a computation, the input neuron has to receive a spike from the environment, which then is immediately sent to the first neuron of the neuron structure simulating the initial instruction p_0.

When the machine halts with the HALT-instruction, we still have to output the result as a spike train. As the first register is never subtracted from during the simulation of the register machine, we can construct the output structure as shown in Figure 8:

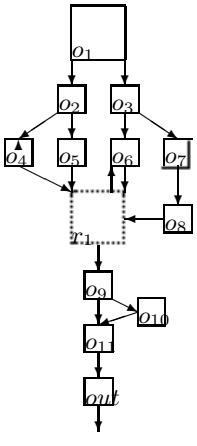


Fig. 8. The output structure (without dummy neurons)

The basic mechanism of the output structure is as follows: the system spikes if and only if there was no spike from the current subtraction from the register or from the last subtraction from the register. This is the case at the beginning (where there is no preceding subtraction) and at the end (when the last subtraction cannot be executed, as the register is empty). The number of steps between these two spikes is always the number in the register minus one (or, if the register was empty, there would be only one spike). Therefore, we need to increment the register (by adding two spikes) before starting the output.

The output of the spiking neural P system (given by the spike train) is equivalent to the output of the corresponding register machine (given by the value in the first register upon halting). This observation concludes the proof. □

Corollary 1. $NSN_{acc}P_*(unbound_3) = NSN_{gen}P_*(unbound_3) = NRE$.

4 Conclusions

By using the characterization of types (where two neurons are of the same type if the rules, the initial configuration and the number of outgoing synapses are identical), we were able to prove that for obtaining computational completeness only two types of neurons are needed, where only the second type of neurons contains unbounded rules. We also showed that three neurons of the second type are enough, which already is the minimal number needed anyway as registers can only be represented by neurons with unbounded rules. However, the number of neurons of the first type could not be bounded; this can be achieved by simulating universal register machines (yet then we have to use spike trains with three spikes to introduce the code of the machine to be simulated as well as its input). Further interesting questions for future research remain, for example, which ingredients are needed to generalize the results for recursively enumerable sets of vectors of natural numbers or which changes in the constructions of the proofs elaborated in this paper are necessary for the input (in the accepting case) or the output (in the generating case) being initially (finally) given as contents of an unbounded input (output) neuron.

References

1. García-Arnau, M., Peréz, D., Rodríguez-Patón, A., Sosík, P.: Spiking neural P systems: Stronger normal forms. In: Gutiérrez-Naranjo, M.A., Păun, G., Romero-Jiménez, A., Riscos-Núñez, A. (eds.) Proceedings of the Fifth Brainstorming Week on Membrane Computing, pp. 33–62 (2007)
2. Ibarra, O.H., Păun, A., Păun, G., Rodríguez-Patón, A., Sosík, P., Woodworth, S.: Normal forms for spiking neural P systems. Theor. Comput. Sci. 372(2-3), 196–217 (2007)
3. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundam. Inf. 71(2,3), 279–308 (2006)
4. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Inc., Upper Saddle River (1967)
5. Pan, L., Păun, G.: Spiking neural P systems: An improved normal form. Theoretical Computer Science 411(6), 906–918 (2010)
6. Păun, A., Păun, G.: Small universal spiking neural P systems. Biosystems 90(1), 48–60 (2007)
7. Păun, G.: Computing with membranes. J. of Computer and System Sci. 61(1), 108–143 (2000)
8. Păun, G.: Membrane Computing: An Introduction. Springer, New York (2002)
9. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Word, Language, Grammar, vol. 1. Springer, Heidelberg (1997)
10. Zeng, X., Zhang, X., Pan, L.: Homogeneous spiking neural P systems. Fundam. Inf. 97(1-2), 275–294 (2009)
11. Zhang, X., Zeng, X., Pan, L.: Smaller universal spiking neural P systems. Fundam. Inf. 87(1), 117–136 (2008)
12. The P Systems Web Page, http://ppage.psystems.eu

P Systems and Unique-Sum Sets

Pierluigi Frisco

School of Math. and Comp. Sciences
Heriot-Watt University
EH14 4AS Edinburgh, UK
P.Frisco@hw.ac.uk

Abstract. We study P systems with symport/antiport and a new model of purely catalytic P systems, called *purely multi-catalytic P systems*, when these devices use only one symbol. Our proofs use unique-sum sets, sets of integer numbers whose sum can only be obtained in a unique way with the elements of the set itself.

We improve some results related to the descriptional complexity of the P systems with symport/antiport considered by us and we define one infinite hierarchy of computations.

1 Introduction

Membrane systems (*P systems*) are an abstract model of computation inspired by the compartmentalisation present in eukariotic cells [14,15,6,3]. For several models of P systems it is possible to define rewriting systems without any compartment able to simulate (that is, mimic the behaviour of) the original P system. This is possible through the use in the rewriting systems of several symbols encoding the compartments of the P system and the passage of the symbols in the P system from one compartment to another.

This possibility to remove the compartments from a P system is something well known in this field of research and it has been formally studied through the use of Petri nets [6,17].

Some models of P systems do not allow a reduction to rewriting systems. If, for instance, a P system with several compartments is ought to have only one symbol, then no other symbol can be introduced in a rewriting system simulating the P system. So, it makes a lot of sense to study P systems with restrictions in the number of used symbols, as these systems are the ones that do need the compartments in order to operate.

The computational power of *P systems with symport/antiport*, one of the most elegant and the most studied model of P system [13], has been studied when restrictions on the number of used symbols are in place [16,1,2,6,11].

Here we continue this line of research on P systems with symport/antiport proving that when these devices operate under maximal parallelism, then the model using only one symbol and with $2n + 1, n \geq 2$, compartments can simulate register machines. In this way we partially answer suggestion for research 5.3 in [6] and we solve a problem stated in [11]. We also prove that when the

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 208–225, 2010.

number of occurrences of the unique symbol is bounded, then an infinite hierarchy on the number of compartments is present. We also study *purely multicatalytic P systems* a model of *catalytic P systems* introduced in the present paper, when these systems have several catalysts and only one symbol which is not a catalyst.

The provided proofs employ the use of *unique-sum sets*: sets of integer numbers whose sum can only be obtained in a unique way with the elements of the set itself.

2 Basic Definitions

We assume the reader to have familiarity with basic concepts of formal language theory [9], register machines and P systems [6,3]. In this section we recall particular aspects relevant to our presentation.

The symbol $\mathbb{N}$ denotes the set of natural numbers $\{0, 1, 2, \ldots\}$, while $\mathbb{N}_+$ denotes $\{1, 2, 3, \ldots\}$ and $\emptyset$ denotes the empty set.

Given a set V its *cardinality*, that is, the number of elements in V, is denoted with $|V|$; V^* denotes the free monoid generated by V with the operation of concatenation, λ indicates the empty word.

Let a grammar $G = (N, T, S, P)$ be of type-0 or of type-3. The length of the strings over T which can be obtained by derivations of G is the *set of numbers generated* by G. The respective classes of numbers are denoted by $\mathbb{N}\,\mathsf{REG}$ and $\mathbb{N}\,\mathsf{RE}$.

For $k \in \mathbb{N}$, $\mathbb{N}_k\,\mathsf{RE}$ equals the family of recursively enumerable sets with elements greater than or equal to k i.e., $\{L \in \mathbb{N}\,\mathsf{RE} \mid \{0, \ldots, k-1\} \cap L = \emptyset\}$, or equivalently, $\{k+L \mid L \in \mathbb{N}\,\mathsf{RE}\}$, where $k+L = \{k+n \mid n \in L\}$. From the point of computational completeness, the families $\mathbb{N}\,\mathsf{RE}$ and $\mathbb{N}_k\,\mathsf{RE}$ are equivalent, as a Turing machine can make the translation, but here we inherit the language definition used in the literature of P systems and make this distinction.

A *multiset* (over V) is a function $M : V \to \mathbb{N} \cup \{+\infty\}$; for $a \in V$, $M(a)$ defines the *multiplicity* of a in the multiset M. We say that an element a of a multiset M has *infinite multiplicity* if $M(a) = +\infty$. In case the multiplicity of an element of a multiset is 1 we indicate just the element, otherwise $(a, M(a))$ is indicated. The *support* of a multiset M is the set $supp(M) = \{a \in V \mid M(a) > 0\}$. The *size* of a multiset is defined by the function $|\,| : (V \to \mathbb{N} \cup +\{\infty\}) \to \mathbb{N} \cup \{+\infty\}$, where for M multiset over V, $|M| = \sum_{a \in supp(M)} M(a)$. The symbol ϕ indicates the *empty multiset*, that is the multiset whose support is the empty set.

Let $M_1, M_2 : V \to \mathbb{N} \cup \{+\infty\}$ be two multisets. The *union* of M_1 and M_2 is the multiset $M_1 \cup M_2 : V \to \mathbb{N} \cup +\{\infty\}$ defined by $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$, for all $a \in V$. The *difference* $M_1 - M_2$ is here defined only when M_2 is included in M_1 (which means that $M_1(a) \geq M_2(a)$ for all $a \in V$) and it is the multiset $M_1 - M_2 : V \to \mathbb{N} \cup \{+\infty\}$ given by $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$. Of course, if $M_1(a) = +\infty$ and $M_2(a)$ is finite, then $M_1(a) - M_2(a) = +\infty$.

In this paper we describe formal systems simulating other formal systems. The concept of simulation is widely used in (theoretical) computer science and it refers to the fact that a device can mimic the behaviour of another device. In the present paper we use the definition of this concept as given in [7].

2.1 P Systems

In this section we introduce the models of P system used in the present paper. Before doing so we define *cell-trees*.

A directed graph $\mu = (N, E)$, where N is the finite set of *vertices* and E is the set of *edges*, is said to be a *cell-tree* if:

- $0 \in N$ where the vertex 0 is the *root* of the tree;
- each vertex in N except the root defines a *membrane compartment* (in this paper referred simply as *compartment*) of a P system Π; the root 0 defines the *environment*;
- the root has only one child; this last is called *skin compartment*;
- the set E is the 'father of' relation present in μ equivalent to the nesting of membranes normally used in the literature of P systems.

In the following we represent cell-tree as boxes with a subscript, the number of the vertex represented by them, and eventually containing other boxes and symbols.

An *accepting P system with symport/antiport* [13,6,3] of *degree* m, $m \geq 1$, is a construct

$$\Pi = (V, \mu, L_0, L_1, \ldots, L_m, R_1, \ldots, R_m, comp)$$

where:

- V is a finite set of symbols;
- $\mu = (N, E)$ is a cell-tree with m vertices underlying Π;
- $L_i, 0 \leq i \leq m$, are multisets over V defining the initial multisets of symbols. All the symbols in L_0 have infinite multiplicity while the ones in $L_1, \ldots, L_m$ do not;
- $R_i, 1 \leq i \leq m$, are sets containing a finite number of *rules* of the form: $(v; \text{in})$, $(v; \text{out})$ (called *symport* rules), or $(w; \text{out}/v; \text{in})$ (called *antiport* rules), with v, w nonempty multisets over V with a finite support. Thus the symports $(\phi; \text{in})$ and $(\phi; \text{out})$ and the antiports $(b; \text{out}/a; \text{in})$ with $a = \phi$ or $b = \phi$ are not allowed;
- $comp \subset \{1, \ldots, m\}$ is the set of *initial compartments*. It is common practice to have $|comp| = 1$, anyhow, in this paper we consider also cases in which $|comp| > 1$.

A *configuration* of a P system with symport/antiport of degree m is given by the $m + 1$-tuple $(M_0 - L_0, M_1, \ldots, M_m)$ of multisets over V associated to the environment and the compartments $\{1, \ldots, m\}$ respectively. Note that the configuration does not record the symbols in the environment that occur with

infinite multiplicity as they are invariant to any configuration. The $m+1$-tuple $(\phi, L_1, \ldots, L_m)$ is called *initial configuration*. For two configurations $(M_0 - L_0, M_1, \ldots, M_m), (M'_0 - L_0, M'_1, \ldots, M'_m)$ of Π we write $(M_0 - L_0, M_1, \ldots, M_m) \Rightarrow (M'_0 - L_0, M'_1, \ldots, M'_m)$ indicating a *transition* from $(M_0 - L_0, M_1, \ldots, M_m)$ to $(M'_0 - L_0, M'_1, \ldots, M'_m)$ that is the application of a set of rules associated to each compartment under the requirement of *maximal parallelism*: rules are applied in parallel to the maximum degree possible. If more than one maximal set of rules can be applied, then exactly one of them is nondeterministically chosen; all rules present in this set are applied in parallel.

The rules R_q associated to a compartment $q \in N \backslash \{0\}$ can change the multisets M_q and M_p of p father of q in μ in the following way:

- a multiset v included in M_p may be subtracted from M_p and may be united to M_q, if the symport rule $(v; \text{in})$ is present in R_p. In this case the multisets change from M_p and M_q to $M'_p = M_p - v$ and $M'_q = M_q \cup v$ respectively;
- a multiset v included in M_q may be subtracted from M_q and united to M_p if the symport rule $(v; \text{out})$ is present in R_q. The multisets change from M_p and M_q to $M'_p = M_p \cup v$ and $M'_q = M_q - v$ respectively;
- a multiset v included in M_p may be united to M_q while, at the same time, a multiset w included in M_q may be united to M_p if the antiport rule $(w; \text{out}/v; \text{in})$ is present in R_q. In this case the multisets of symbols change from M_p and M_q to $M'_p = (M_p - v) \cup w$ and $M'_q = (M_q - w) \cup v$ respectively.

In general, if a multiset v is subtracted from M_p and united to M_q we say that v *passes* from compartment p to compartment q.

The *weight* of a rule is given by $|v|$ (that is, the cardinality of the multiset v) in case of a symport $(v; \text{in})$ or $(v; \text{out})$ and by $max(\{|v|, |w|\})$ in case of an antiport $(v; \text{out}/w; \text{in})$.

A *computation* is a finite sequence of transitions between configurations of a system Π starting from the initial configuration $(\phi, L_1, \ldots, L_m)$. If a computation is finite, then the last configuration is called *final* and we say that the system *halts*.

The result of the computation is given by the vector of multiset of symbols (the vector has dimension $|comp|$, if this dimension is 1, then we speak of set of numbers) present in the compartments in *comp* in the initial configuration when, on such initial configuration, the P system halts (that is, when the multiset of applicable rules has empty support).

The set of numbers whose elements are bigger than $k, k \in \mathbb{N}_+$, accepted by P systems with symport/antiport operating under maximal parallelism using s symbols, degree at most m, using symports of weight at most p and antiports of weight at most q is denoted by $\mathbb{N}_k \mathsf{aO}_s \mathsf{P}_m(\text{sym}_p, \text{anti}_q)$. When p, q or m are not bounded, then they are replaced by $*$.

We now introduce a model of purely catalytic P systems [14,6,3], called *purely multi-catalytic P systems*, with limitations in its alphabet and having a new kind of rules.

An *accepting purely multi-catalytic P system* of *degree* m is a construct

$$\Pi = (V, C, \mu, L_1, \ldots, L_m, R_1, \ldots, R_m, comp)$$

where:

$V = \{a\}$ is an alphabet;
$C, C \cap V = \emptyset$, is a set of *catalysts*;
$\mu = (N, E)$ is a cell-tree with m vertices underlying Π;
$L_i, 1 \leq i \leq m$, are multisets over V defining the initial multisets of symbols;
$R_i, 1 \leq i \leq m$, are finite sets of *rules* of the kind: $ca^p \rightarrow c(a^q, tar)$ with $p, q \in \mathbb{N}_+$, $c \in C$ and $tar \in \{here, in, out\}$, where the set $\{here, in, out\}$ contains *target indicators*;
$comp \in \{1, \ldots, m\}$ is the *initial compartment*.

A *configuration* of Π is an m-tuple $(M_1, \ldots, M_m)$ of multisets over V associated with the compartments of Π. The m-tuple $(L_1, \ldots, L_m)$ is called *initial configuration*.

Let $\Pi = (V, C, \mu, L_1, \ldots, L_m, R_1, \ldots, R_m, comp)$ be an accepting purely multi-catalytic P system and let j, i and k be three vertices in μ such that i is the parent of j and j is the parent of k. Moreover, let R_j be the set of rules associated with compartment j and let M_j, M_i and M_k be multisets over V, associated with vertices j, i and k, respectively, such that $ca^p \rightarrow c(a^q, tar) \in R_j, p, q \in \mathbb{N}_+$ and $tar \in \{here, in, out\}$. Depending on the value of tar the application of $ca^p \rightarrow c(a^q, tar) \in R_j$ changes the multisets M_j, M_i and M_k according to the following:

if $tar = here$, then M_j becomes $M'_j = M_j \setminus \{a^p\} \cup \{a^q\}$ while M_i and M_k remain unchanged;
if $tar = in$, then M_j becomes $M'_j = M_j \setminus \{a^p\}$, M_k becomes $M'_k = M_k \cup \{a^q\}$ and M_i remains unchanged;
if $tar = out$, then M_j becomes $M'_j = M_j \setminus \{a^p\}$, M_i becomes $M'_i = M_i \cup \{a^q\}$ and M_k remains unchanged.

In the following the target indicator *here* is omitted. For two configurations $(M_1, \ldots, M_m)$ and $(M'_1, \ldots, M'_m)$ of Π we write $(M_1, \ldots, M_m) \Rightarrow (M'_1, \ldots, M'_m)$ to denote a *transition* from $(M_1, \ldots, M_m)$ to $(M'_1, \ldots, M'_m)$, that is, the application of a multiset of rules associated with each compartment under the requirement of maximal parallelism.

A *computation* is a sequence of transitions between configurations of a system Π starting from the initial configuration $(L_1, \ldots, L_m)$. If a computation is finite, then the last configuration is called *final* and we say that the system *halts*.

Given a purely multi-catalytic P system $\Pi = (V, C, \mu, L_1, \ldots, L_m, R_1, \ldots, R_m, comp)$ with $comp = c_1$, we consider two results from a computation of Π:

$\mathsf{N}(\Pi)$: the sum of occurrences of a and catalysts present in *comp* in the initial configuration, when, on such initial configuration, Π halts;
$\mathsf{N}_{-c}(\Pi)$: the occurrences of a present in *comp* in the initial configuration, when, on such initial configuration, Π halts.

Moreover,

$\mathbb{N}_k\mathsf{aO}_1\mathsf{P}_m(\text{pmcat}_p) = \{\mathsf{N}(\Pi) \mid \Pi$ is an accepting purely multi-catalytic P system operating under maximal parallelism with degree at most m and using at most p catalysts and such that each element in $\mathsf{N}(\Pi)$ is bigger than $k, k \in \mathbb{N}_+$;

$\mathbb{N}\mathsf{aO}_1\mathsf{P}_{m,-c}(\text{pmcat}_p) = \{\mathsf{N}_{-c}(\Pi) \mid \Pi$ is an accepting purely multi-catalytic P system operating under maximal parallelism, of degree at most m and using at most p catalysts$\}$;

So, for instance, if such a P system has $comp = 2$ and the only initial configuration for which the P system halts is such that compartment 2 has two catalysts and three occurrences of a, then:

$\{5\}$ is the set accepted if all symbols counted with their multiplicity are considered;

$\{3\}$ is the set accepted if only the a symbols are considered;

2.2 Register Machines

A *register machine* [12] (also known as *(multi)counter machines*, *multipushdown machines*, *program machine* and *counter automata*) with n registers ($n \in \mathbb{N}_+$) is a construct $M = (S, R, s_1, s_f)$, where:

S is a finite set of *states*;

R is a finite set of *instructions* of the form $(s_p, \gamma_t^-, s_q), (s_p, \gamma_t^+, s_q)$ or $(s_p, \gamma_j{}^{=0}, s_q)$, with $s_p, s_q \in S,\ s_p \neq s_f, 1 \leq t \leq n$;

$s_1, s_f \in S$ are respectively called the *initial* and *final* states.

A *configuration* of a register machines M with n registers is given by an element in the $n+1$-tuples $(s,\ \mathbb{N}^n),\ s \in S$. Given two configurations $(s, val(\gamma_1), \ldots, val(\gamma_n))$ and $(s', \gamma_1', \ldots, \gamma_n')$ (where $val : \{\gamma_1, \ldots, \gamma_n\} \to \mathbb{N}$ is the function returning the content of a register) we define a *computational step* as $(s, val(\gamma_1), \ldots, val(\gamma_n)) \vdash (s', \gamma_1', \ldots, \gamma_n')$:

- if $(s, \gamma_t^-, s_q) \in S$ and $val(\gamma_t) \neq 0$, then $s' = s_q,\ \gamma_t' = val(\gamma_t) - 1,\ \gamma_k' = val(\gamma_k),\ k \neq t,\ 1 \leq k \leq n$;
  if $val(\gamma_t) = 0$, then the register machine halts in the non-final state s;
  (informally: in state s if the content of register γ_t is greater than 0, then subtract 1 from that register and change state into s_q, otherwise halt in a non-final state);
- if $(s_p, \gamma_j^{=0}, s_q) \in S$ and $val(\gamma_t) = 0$, then $s' = s_q,\ \gamma_k' = val(\gamma_k),\ 1 \leq k \leq n$;
  if $val(\gamma_t) \neq 0$, then the register machine halts in the non-final state s;
  (informally: in state s if the content of register γ_t is 0, then change state into s_q, otherwise halt in a non-final state);
- if $(s, \gamma_t^+, s_q) \in S$, then $s' = s_q, \gamma_t' = val(\gamma_t) + 1,\ \gamma_k' = val(\gamma_k),\ k \neq t,\ 1 \leq k \leq n$;
  (informally: in state s add 1 to register γ_t and change state into s_q).

A *computation* is a sequence of computational steps of a register machine M starting from the *initial configuration* $(s_1, val(\gamma_1), 0, \ldots, 0)$. If a computation is finite, then the last configuration is called *final*. If a final configuration has s_f as state and all the counters are empty, then we say that M *halts* and it *accepts* the input $val(\gamma_1)$. For this reason γ_1 is called the *input register* and M is called an *accepting register machine*. Starting from an initial configuration $(s_1, val(\gamma_1), 0, \ldots, 0)$ a register machine M could have a finite sequence of computational steps in which the last one does not have s_f as state. In this case we say that M *stops* and $val(\gamma_1)$ is not accepted.

Partially blind register machines [8] are defined as register machines without test on zero. The only allowed operations are (s, γ^+, s') and (s, γ^-, s') where γ is a register. In case the machine tries to subtract from a register having value zero it halts in a non-final state. Partially blind register machines are strictly less powerful from a computational point of view than register machines.

We summarise in the following proposition, results from [12,4]:

Proposition 1. *Register machines with three registers can accept* $\mathbb{N}\,\mathsf{RE}$*, the number of registers can be decreased to two if specific input format (for example,* 2^x *instead of* x*) is used.*

Register machines with only one register can accept $\mathbb{N}\,\mathsf{REG}$.

Restricted register machines are defined as register machines restricted in their operations: they can increase the value of a register, say β, only if they decrease the value of another register, say γ, at the same time.

So, restricted register machines have only one kind of instruction: $(s, \gamma^-, \beta^+, v, w)$ with s, v, w states and γ, β different registers of the restricted register machine. If when in state s the content of register γ can be decreased by 1, then the one of register β is increased by 1 and the machine goes into state v, otherwise no operation is performed on the registers and the machine goes into state w.

Here a theorem from [10] that we need:

Theorem 1. *Restricted register machines with n+1 registers are more powerful from a computational point of view than those with n registers.*

A consequence of this theorem is that an infinite hierarchy is induced, by means of the number of registers, among families of computed vectors of numbers.

2.3 Unique-Sum Sets

Some proof in this paper use the following mathematical concepts.

Definition 1. *Let* $U = \{u_1, \ldots, u_p\}$ *be a set of distinct natural numbers and* $\sigma_U = \sum_{i=1}^{p} u_i$ *the sum of the elements of* U*. The set* U *is said to be a unique-sum set if the equation* $\sum_{i=1}^{p} c_i u_i = \sigma_U, c_i \in \mathbb{N}$*, has only the solutions* $c_i = 1,\ 1 \leq i \leq p$.

An example of unique-sum set is $U' = \{4, 6, 7\}$ as $4+6+7 = 17$ and 17 cannot be obtained with any other linear combination of 4, 6 and 7. The set $U'' = \{4, 5, 6\}$ is not a unique-sum set as $4 + 5 + 6 = 15 = 5 + 5 + 5$.

It should be clear that any subset of a unique-sum set is a unique-sum set, too. In particular none of the elements of a unique-sum set can be obtained as a linear combination of the remaining elements in the set.

Definition 2. *The family of sets M1SV is* $U_p = \cup_{m=1}^{p}(2^p - 2^{p-m})$ *with* $g_p \in \mathbb{N}_+^p, p \in \mathbb{N}_+$.

The sum of the elements of the sets in this family is $\sigma_{U_p} = (p-1)2^p + 1$. The first sets in this family are:

$$U_1 = \{1\};$$
$$U_2 = \{2, 3\};$$
$$U_3 = \{4, 6, 7\};$$
$$U_4 = \{8, 12, 14, 15\};$$
$$U_5 = \{16, 24, 28, 30, 31\};$$
$$U_6 = \{32, 48, 56, 60, 62, 63\}.$$

From [5] it is known that:

Theorem 2. *For all* $p \in \mathbb{N}_+$ *the set* U_p *in M1SV is a unique-sum set.*

3 P Systems with Symport/Antiport

In [11] it was proved that P system with symport/antiport operating under maximal parallelism, with only one symbol and degree $2n + 3$ can simulate a partially blind register machines with n registers. In [11] it was also proved that if priorities are added to the rules, then the obtained P system, having $n + 3$ compartments, can simulate register machines with n registers. The former result was improved in [6] where it was proved that any partially blind register machine with n registers can be simulated by a P system with symport/antiport with only one symbol, degree $n+3$ and operating under maximal parallelism. Here we prove that P systems with symport/antiport operating under maximal parallelism, with only one symbol and degree $2n + 1$ can simulate register machines with n registers.

Theorem 3. *Any accepting register machines with n registers can be simulated by an accepting P system with symport/antiport operating under maximal parallelism, using only one symbol and with degree* $2n + 1$.

Proof. Let $M = (S, I, s_1, s_f)$ be a register machine with n registers $\gamma_1, \ldots, \gamma_n$. We define the P system with symport/antiport operating under maximal parallelism

$$\Pi=(\{a\},\mu,\{a\},L_1,\ldots,L_{n+1},L_{2'},\ldots,L_{n+1'},R_1,\ldots,R_{n+1},R_{2'},\ldots,R_{n+1'},\{2\})$$

where:

$$\begin{aligned}
\mu &= \left[\left[\;\right]_2\left[\;\right]_{2'}\cdots\left[\;\right]_{n+1}\left[\;\right]_{n+1'}\right]_1;\\
L_1 &= a^{c(s_1)+\sum_{i=2}^{n+1}2b(j)};\\
L_{j+1} &= a^{2b(j)+val(\gamma_j)},\ 1\le j\le n;\\
L_{j+1'} &= a^{2b(j)},\ 1\le j\le n;\\
R_1 &= \{1:(a^{c(p)};\mathrm{out}/a^{c^{(1)}(q)+2b(t)+1};\mathrm{in}),2:(a^{c^{(1)}(q)};\mathrm{out}/a^{c^{(2)}(q)};\mathrm{in}),\\
&\quad 3:(a^{c^{(2)}(q)+2b(t)};\mathrm{out}/a^{c(q)};\mathrm{in})\mid(s_p,\gamma_t^+,s_q)\in I\}\cup\\
&\quad \{4:(a^{c(p)};\mathrm{out}/a^{c^{(3)}(q)+2b(t)};\mathrm{in}),5:(a^{c^{(3)}(q)};\mathrm{out}/a^{c^{(4)}(q)};\mathrm{in}),\\
&\quad 6:(a^{c^{(4)}(q)+2b(t)+1};\mathrm{out}/a^{c(q)};\mathrm{in})\mid(s_p,\gamma_t^-,s_q)\in I\}\cup\\
&\quad \{7:(a^{c(p)};\mathrm{out}/a^{c^{(5)}(q)+3b(t)+1};\mathrm{in}),8:(a^{c^{(5)}(q)};\mathrm{out}/a^{c^{(6)}(q)};\mathrm{in}),\\
&\quad 9:(a^{c^{(6)}(q)};\mathrm{out}/a^{c^{(7)}(q)};\mathrm{in}),10:(a^{c^{(7)}(q)};\mathrm{out}/a^{c^{(8)}(q)};\mathrm{in}),\\
&\quad 11:(a^{c^{(8)}(q)};\mathrm{out}/a^{c^{(9)}(q)};\mathrm{in}),12:(a^{c^{(9)}(q)+3b(t)+1};\mathrm{out}/a^{c^{(q)}};\mathrm{in}),\\
&\quad \mid(s_p,\gamma_t^{=0},s_q)\in I\}\cup\\
&\quad \{13:(a^{c(s_f)+\sum_{i=2}^{n+1}2b(j)};\mathrm{out}),14:(a;\mathrm{out}/a^{\sigma+1};\mathrm{in})\};\\
R_{j+1'} &= \{15:(a^{2b(j)};\mathrm{out}/a^{2b(j)};\mathrm{in}),16:(a^{2b(j)};\mathrm{out}/a^{2b(j)+1};\mathrm{in}),\\
&\quad 17:(a^{2b(j)+1};\mathrm{out}/a^{2b(j)};\mathrm{in}),18:(a^{2b(j)};\mathrm{out}/a^{3b(j)+1};\mathrm{in}),\\
&\quad 19:(a^{3b(j)+1};\mathrm{out}/a^{2b(j)};\mathrm{in})\}\qquad 1\le j\le n;\\
R_{j+1} &= \{20:(a^{2b(j)};\mathrm{out}/a^{2b(j)+1};\mathrm{in}),21:(a^{2b(j)+1};\mathrm{out}/a^{2b(j)};\mathrm{in}),\\
&\quad 22:(a;\mathrm{out}/a^{b(j)};\mathrm{in})\}\qquad 1\le j\le n;
\end{aligned}$$

In order to facilitate the explanation rules have been numbered. The number of occurrences of a brought in by rule 14 is $\sigma+1$ where σ is the sum of the occurrences of a brought out by the remaining rules present in compartment 1. The reason for this value for σ is explained in the following.

This proof requires the use of a unique-sum U with at most $n+10|S|$ elements. Different multiplicities of a, where the multiplicities are elements in U, are associated with each of the registers and instructions of M. This is performed by the function $b:\{1,\ldots,n\}\to U$ and the ten functions c, $c^{(1)},\ldots,c^{(9)}$ all from S to U, injective and with disjoint values. The exact definition of these functions is irrelevant for the proof.

The simulation performed by the P system Π is strongly based on the use of a unique-sum set and on the property that for such sets none of the elements can be obtained as a linear combination of the remaining elements in the set. During the computation of Π, different occurrences of the symbol a are present in the skin compartment. Specific sequences of applied rules allow Π to simulate instructions of M. Other sequences of applied rules let Π to never halt.

The compartments $j+1, j+1'$, $1\le j\le n$, are uniquely associated with registers in M. Each of these compartments contains at least $b(j)$ occurrences of a in the initial configuration. This number of occurrences of a represents 0 as content of the registers in M. The addition of 1 to register γ_j in M is performed adding one occurrence of a to compartment $j+1$. Conversely for the

subtraction. The presence of just $a^{c(s)},\ s \in S$, in the skin compartment indicates that Π simulates the register machine being in state s.

The application of rule 14 let Π start an infinite computation. This is due to the fact that maximal parallelism is used and to the fact that even if all remaining rules in compartment 1 were used (bringing out σ occurrences of a), then there would still be another occurrences of s to be used by rule 14.

In the following description it will be clear that the symbols present in the skin compartments and in compartments $j+1', 1 \leq j \leq n$, are always kept busy, that is, they are always subject to a rule. In particular, rule 15 moves $2b(j)$ occurrences of a between the skin and compartments $j+1', 1 \leq j \leq n$.

Rules in parenthesis denote their parallel application.

The simulation of instructions of the kind (s_p, γ_t^+, s_q) is performed by the sequential application of rules (1,15), (2, 16, 15), (3, 20, 15). It should be clear that when (1, 15) takes place, rule 15 is applied to all $j+1'$ compartments, while in the following two steps rule 15 is applied to all the $j+1'$ compartments with the exception of compartment $t+1$.

If the sequence of rules (1, 15), (2, 16, 15), (3, 15) is instead applied, then 1 occurrences of a remains in compartment $t+1$, and this leads Π to never halt. This occurs because we assume that M halts with all registers empty, so, sooner or later Π tries to subtract 1 from compartment $t+1$ and this would lead Π to never halt.

The simulation of instructions of the kind (s_p, γ_t^-, s_q), if in compartment $t+1$ there are at least $2b(t)+1$ occurrences of a, is performed by the sequential application of rules (4, 21, 15), (5, 17, 15), (6, 15). It should be clear that when (6, 15) takes place, rule 15 is applied to all $j+1'$ compartments, while in the other two steps rule 15 is applied to all the $j+1'$ compartment with the exception of compartment $t+1$.

If in compartment $t+1$ there are less than $2b(t)+1$ occurrences of a, then rule 21 cannot be applied and, as a consequence, rule 6 cannot be applied. The subsequent application of rule 14 let Π to never halt.

The simulation of instructions of the kind $(s_p, \gamma_t^{=0}, s_q)$, if in compartment $t+1$ there are just $2b(t)$ occurrences of a (simulating the counter t being empty), is performed by the sequential application of rules (7,15), (8, 18, 15), (9, 20, 15), (10, 21, 15), (11, 19, 15), (12, 15). It should be clear that when (7, 15) and (12, 15) take place, rule 15 is applied to all $j+1'$ compartments, while in the other steps rule 15 is applied to all the $j+1'$ compartment with the exception of compartment $t+1$.

If compartment $t+1$ contains more than $2b(t)$ occurrences of a, then the following rules are applied: (7,15), (8, 18, 22, 15). The application of rule 22 let $b(t)$ occurrences of a to be brought into compartment $t+1$. As a consequence of this, rule 19 and rule 12 cannot be applied, rule 14 is applied and Π never halts.

When $a^{c(s_f)}$ is present in compartment 1, then the application of rule 30 let $a^{c(s_f)+\sum_{i=2}^{n+1} 2b(j)}$ to pass to the environment, rule 15 is no longer applied, and the computation halts only if rule 14 has never been applied.

The simulation of M is faithful, that is, Π cannot simulate sequences of instructions that cannot be performed by M. This means that if Π starts in a configuration in which $a^{c(s_1)+\sum_{i=2}^{n+1} 2b(j)}$ is present in the skin compartment, $a^{2b(1)}$ is present in compartment $2'$, $a^{2b(1)+val(\gamma_1)}$ is present in compartment 2 and compartments $j+1, j+1', 3 \leq j \leq n$, contain $a^{2b(j)}$, then Π halts with $a^{2b(j)}$ in compartments $j+1, j+1', 1 \leq j \leq n$, only if M reached the final state with all registers empty if it started from the configuration $(s_1, val(\gamma_1), 0, \ldots, 0)$. □

From the previous theorem and Proposition 1 we have:

Corollary 1. *There exist* $b, b', p, p', q, q' \in \mathbb{N}_+$ *such that*

$\mathbb{N}_b\mathsf{O}_1\mathsf{P}_7(sym_p, anti_q) = \mathbb{N}_b\mathsf{RE}$;

$\mathbb{N}_{b'}\mathsf{O}_1\mathsf{P}_5(sym_{p'}, anti_{q'}) = \mathbb{N}_{b'}\mathsf{RE}$ *if a specific input format is used,*

If a P system with symport/antiport Π is such that the skin compartment contains no rule then, for each computation of Π, the number and type of symbols in Π is constant and equal to the one of the initial configuration. In the following we consider such systems when they use only one symbol.

Theorem 4. *The set of vectors accepted by restricted register machines coincides with the ones accepted by P systems with symport/antiport operating under maximal parallelism, using a constant number of occurrences of only one symbol.*

Proof. Part I: (These P systems can simulate restricted register machines) This proofs follows from the one of Theorem 3 after a few changes in the P system there. Let $M = (S, I, s_1, s_f)$ be a restricted register machines with n registers $\gamma_1, \ldots, \gamma_n$.

We define the P system with symport/antiport operating under maximal parallelism

$\Pi = (\{a\}, \mu, \{a\}, L_1, \ldots, L_{n+3}, L_{2'}, \ldots, L_{n+2'}, R_1, \ldots, R_{n+3}, R_{2'}, \ldots, R_{n+2'}, \{2, \ldots, n+1\})$

where:

$\mu = [\,[\,]_2 \cdots [\,]_{n+1}\, [\,]_{n+2}\,]_1$;

$L_{n+2} = a$;
$L_{n+2'} = \phi$;
$L_{n+3} = a^\sigma$;
$R_{n+2} = (a; \text{out}/a; \text{in})$;
$R_{n+2'} = (a; \text{in})$;
$R_{n+3} = \emptyset$;
where:

rule 14 in the proof of Theorem 3 is absent in the present system;
σ is the sum of the unique sum set U used in the proof of Theorem 3;
the remaining elements of Π are defined as in the proof of Theorem 3;
the unique sum set U and the functions $b, c, c^{(1)}, \ldots, c^{(9)}$ are as defined in the proof of Theorem 3.

The fact that the number of occurrences of the only symbol is constant in Π derives from the fact that in restricted register machines the sum of the content of the counters is constant.

The simulation of the instructions $(s, \gamma^-, \beta^+, v, w)$ can be performed with the non-deterministic simulations of either $(s, \gamma^{=0}, w)$ or (s, γ^-, v') and (v', β^+, v), as explained in the proof of Theorem 3, where v' is a newly introduced state uniquely associated to $(s, \gamma^-, \beta^+, v, w)$.

It should be clear that, as the sum of the registers in M is constant, in Π occurrences of a needed to simulate the instructions of M are provided by compartment L_{n+3}. Moreover, if M starts in a configuration with $val(\gamma_1), \ldots, val(\gamma_n)$ in its registers, then Π starts in an initial configuration with $2b(j) + var(\gamma_j)$ in compartment $j+1, 1 \leq j \leq n$. So, if M reaches its final state s_f on input $val(\gamma_1), \ldots, val(\gamma_n)$, then Π could halt on input $2b(j) + var(\gamma_j), 1 \leq j \leq n$, accepting this vector of numbers.

The infinite loop primed by rule 14 in the proof of Theorem 3, is here replaced by compartments $n+2, n+2'$ and the symbols and rules associated to them. If the rule in compartment $n+2'$ is applied, then the rule in compartment $n+2$ is continuously applied and Π never halts.

If the simulated restricted register machine has n registers, then the simulating P system has $2n+4$ compartments.

Part II: (Restricted register machines can simulate these P systems) Let Π be such a P system and let M be the restricted register machine simulating it. Each compartment of Π is associated to a different register in M whose content reflects the number of occurrences of the only symbol in the associated compartment. The simulations of each rule of Π is performed by a sequence of instructions decreasing and increasing the value of registers. With 'sequence of instructions' we mean instructions such that the output state of one instruction is the input state of the following instruction in the sequence.

Let, for instance, compartment s contain compartment r and let $(a^p; \text{out}/a^q; \text{in})$ belong to the set of rules associated to r. The set of instructions of M will then contain a sequence of p instructions decreasing the value of register r and increasing the one of register s followed by a sequence of q instructions decreasing the value of register s and increasing the one of register r. If the simulation of the whole rule cannot be performed, then M restores the content of the registers (this can be done keeping track in the finite states of how far the simulation of a rule went) and M goes on trying to simulate another rule. The machine M tries to simulate all the rules in a random sequence (all the possible sequences can be encoded in the finite state control of M) and, a rule is simulated as many time as possible (this is because Π works under maximal parallelism). If in one cycle none of the rules could be simulated (again, the finite number of states can keep track of this), then M goes into its final state. □

We do not know if $2n+4$ is the minimum number of compartments needed for the P systems in Theorem 4 to simulate restricted register machines with n registers. Anyhow, as these P systems have a constant number of only one symbols, it is

unlikely that such a P system with a number of compartment independent from n could simulate restricted register machines with an arbitrary number n of registers.

So, knowing Theorem 4 and that restricted register machines induce an infinite hierarchy on the number of registers, we feel confident to say that:

Corollary 2. *P systems with symport/antiport operating under maximal parallelism, using a constant number of occurrences of only one symbol induce an infinite hierarchy on the number of compartments.*

4 Purely Multi-catalytic P Systems

In this section we show how unique-sum sets can be useful to purely catalytic P systems using only one symbol.

Theorem 5. *Any accepting register machines with n registers can be simulated by an accepting purely multi-catalytic P system operating under maximal parallelism, using only one symbol and with degree $2n+1$.*

Proof. Let $M = (S, I, s_1, s_f)$ be an accepting register machine with n registers: $\gamma_1, \ldots, \gamma_n$. We define the purely multi-catalytic P system

$$\Pi = (\{a\}, C, \mu, L_2, R_1, \ldots, R_{2n+1}, 2)$$

where:

$C = \{c_1\} \cup \{c_j, c_{j'}, c_{j'}^{(1)}, c_{j'}^{(2)} \mid 2 \le j \le n+1\}$;

$\mu = [\,[\,]_2 \cdots [\,]_{n+1}\,]_1$;

$L_1 = a^{f(s_1)}$;

$L_2 = a^{2val(\gamma_1)}$;

$L_{j+1} = \phi,\ 2 \le j \le n$;

$L'_{j+1} = \phi,\ 1 \le j \le n$;

$R_1 = \{1 : c_1 a^{f(p)} \to c_1 a^{f(q)}(a^2, in_{t'+1}) \mid (p, \gamma_t^+, q) \in I\} \cup$
$\{2 : c_1 a^{f(p)} \to c_1 a^{f^{(1)}(q)}(a^{b(t)}, in_{t'+1}),$
$3 : c_1 a^{f^{(1)}(q)} \to c_1 a^{f^{(2)}(q)},$
$4 : c_1 a^{f^{(2)}(q)+b(t)+2} \to c_1 a^{f(q)},$
$5 : c_1 a^{f(p)} \to c_1 a^{f^{(3)}(p)},$
$6 : c_1 a^{f^{(3)}(p)} \to c_1 a^{f^{(1)}(q)}(a^{b(t)}, in_{t'+1}) \mid (p, \gamma_t^-, q) \in I\} \cup$
$7 : \{c_1 a^{f(p)} \to c_1 a^{f^{(4)}(q)}(a^{b(t)}, in_{t'+1}),$
$8 : c_1 a^{f^{(4)}(q)} \to c_1 a^{f^{(5)}(q)}(a^{b(t)}, in_{t'+1}),$
$9 : c_1 a^{f^{(5)}(q)+b(t)+2} \to c_1 a^{f^{(6)}(q)}(b(t), in_{t'+1}),$
$10 : c_1 a^{f^{(6)}(q)+b(t)+2} \to c_1 a^{f^{(7)}(q)}(b(t), in_{t'+1}),$
$11 : c_1 a^{f^{(7)}(q)+b(t)+2} \to c_1 a^{f^{(8)}(q)},$

$$
\begin{aligned}
& 12 : c_1 a^{f^{(8)}(q)+b(t)+2} \rightarrow c_1 a^{f(q)} \mid (p, \gamma_t^{=0}, q) \in I\} \cup \\
& \{13 : c_1 a^{f(s_f)} \rightarrow c_1 (a^{f(s_f)}, out), 14 : c_1' a \rightarrow c_1' a^{\sigma+1}\}; \\
R_{j'} = \{ & 17 : c_{j'} a^2 \rightarrow c_{j'} (a^2, in_j), \\
& 18 : c_{j'} a^{b(j-1)+2} \rightarrow c_{j'} (a^{b(j-1)+2}, out), \\
& 19 : c_{j'}^{(1)} a^{b(j-1)} \rightarrow c_{j'}^{(1)} (a^{b(j-1)}, out), \\
& 20 : c_{j'}^{(1)} a^2 \rightarrow c_{j'}^{(1)} (a^2, in_j), \\
& 21 : c_{j'}^{(1)} a^{b(j-1)-2} \rightarrow c_{j'}^{(1)} (a^{b(j-1)-2}, out), \\
& 22 : c_{j'}^{(2)} a^2 \rightarrow c_{j'}^{(2)} (a^2, out)\} \qquad 2 \leq j \leq n+1; \\
R_j = \{ & 23 : c_j a^2 \rightarrow c_j (a^2, out)\} \qquad 2 \leq j \leq n+1.
\end{aligned}
$$

In order to facilitate the explanation rules have been numbered. The number of occurrences of a generated by rule 14 is $\sigma+1$ where σ is the sum of occurrences of a used by the remaining rewriting rules in R_1.

This proof requires the use of a unique-sum set U with at least $n + 9|S|$ elements. Different multiplicities of a, where the multiplicities are elements in U, are associated with each of the registers and instructions of M. This is performed by the function $b : \{1, \ldots, n\} \rightarrow U$ and the nine functions $f, f^{(1)}, \ldots, f^{(8)}$ all from S to U, all injective and with disjoint values. The exact definition of these functions is irrelevant for the proof, the only thing that is essential is that the different elements of U are at least 3 units apart from each other. The reason for this requirement is explained in the following.

The simulation performed by the P system Π is strongly based on the use of a unique-sum set and on the property that for such sets none of the elements can be obtained as a linear combination of the remaining elements in the set. During the computation of Π, different occurrences of the symbol a are present in the skin compartment. Specific sequences of applied rules allow Π to simulate instructions of M. Other sequences of applied rules let Π to never halt.

If rule 14 is applied, then an infinite loop, given by the repeated application of rule 14. If this happens, then Π never halts.

The compartments $2 \leq j, j' \leq n+1$, in Π are associated in a unique way to the registers of M. If during a simulation, register γ_j contains the value $val(\gamma_j)$, then the sum of the occurrences of the symbol a present in j and j' is $2val(\gamma_j)$. The addition of 1 to register γ_j is then simulated with the addition of two occurrences of a to compartment j'. The subtraction of 1 from register γ_j is simulated with the removal of two occurrences of a from compartment j'.

During the simulation, if compartment j' contains at least 2 occurrences of a, then either rule 17 or rule 20 can be applied. The passage of a^2 from compartment j to compartment j' is performed by the application of rule 23. This means that if both compartments j and j' contain at least 2 occurrences of a, then these occurrences keep been moved to j' and j, respectively. In the following description we avoid to repeat that the application of these rules can take place during the simulation of the instructions of M.

If the skin compartment has $f(s)$ occurrences of a, then Π simulates M being in state s.

The simulation of instructions of the kind (p, γ_t^+, q) is performed by the sequential application of rule 1 and then either rule 17 or 20. In this way the number of occurrences of a in the skin compartment changes from $f(p)$ to $f(q)$, indicating that Π simulates the state change of M from p to q, and 2 occurrences of a pass to compartment t' and then to compartment t, simulating the addition of 1 unit to register γ_t.

To describe the simulation of instructions of the kind (p, γ_t^-, q) we have to consider different configurations of t and t' when rule 2 is applied:

i) a^2 is present in t' and $a^k, k \geq 2, k$ even, is present in t;
ii) a^2 is present in t' and t does not contain any a;
iii) t' does not contain any a and t contains a^2.

Because of rules 17, 20 and 23, it cannot be that when t' does not contain any a, t contains more than 2 occurrences of a.

In the following rule numbers in parenthesis denote that the rules are applied in parallel. The simulation of such instructions can non-deterministically start with either rule 2 or rule 5. If it starts with rule 2 and Π is in configuration *ii*, then the applied rules are (2, 17) (or (2, 18)) followed by (3, 19, 23) (or (3, 17, 20, 22, 23), or (3, 17, 21, 23)). In all these cases, rule 4 cannot be subsequently applied and Π starts an infinite loop. Similarly, if the simulation starts with rule 5 and Π is in configuration *iii*. In the other cases the simulation of the instruction can be completed.

Now we describe these possibilities one by one.

If Π is in configuration *i*, then (2, 17, 23) can be applied. When this happens compartment t' has $b(t)+2$ occurrences of a. If now rules (3, 18, 23) are applied, then in the following configuration rules (4, 17), and eventually also 23, can be applied. This is a simulation of (p, γ_t^-, q).

When $a^{b(t)+2}$ is present in t', then other groups of rules can be applied in parallel. It is important to notice that maximal parallelism forces all these $b(t)+2$ occurrences of a to be subject to a rule. For instance, it can be that (3, 17, 19, 23) are applied. If this occurs, then, as rule 3 cannot be applied, rule 14 is applied starting in this way an infinite loop.

If Π is in configuration *iii*, then (2, 23) can be applied. What can happen next is similar to what just described.

If Π is in configuration *ii*, then rules (2, 17) are applied. When this happens $b(t)$ occurrences of a are present in t' and the applied rules let Π start an infinite loop.

If Π is in configuration *i*, then (5, 17, 23), (6, 17, 23) can be applied. When this happens compartment t' has $b(t) + 2$ occurrences of a and the simulation can go on as described in the above.

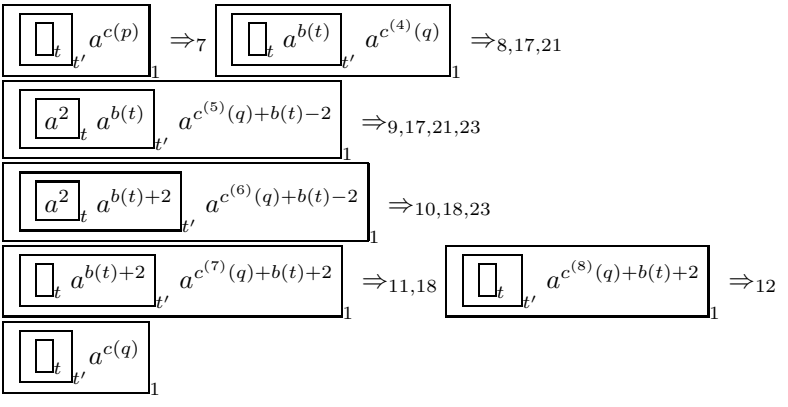
If Π is in configuration *ii*, then rules (5, 17), (6, 23) can be applied, again $b(t)+2$ occurrences of a and the simulation can go on as described in the above.

If Π is in configuration *iii*, then rules (5, 23) can be applied. When this happens $b(t)$ occurrences of a are present in t' and the applied rules let Π start an infinite loop.

The simulation of instructions of the kind $(p, \gamma_t^{=0}, q)$ can start while t and t' are in one of the following configurations:

iv) a^2 is present in t' and at least a^2 is present in t;
v) a^2 is present in t' and t does not contain any a;
vi) t' does not contain any a and t contains a^2;
vii) both t and t' do not contain any a.

Only configuration *iv* can let Π not to end up in an infinite loop. We describe this simulation with a detailed description using a graphical representation of the compartments in Π. Here we indicate only the compartments and symbols relevant for the considered explanation.



In the above simulation many other sequences of configurations could occur but none of them would lead the system to a halt. For instance, instead of applying rules (8, 17, 21) the rules (8, 19) or (8, 17, 20, 22) could be applied but in both cases the computation would go on forever.

The reason why we ask that the elements of U are at least 3 units apart from each other is that because some configurations see 'spare' a's present in the skin compartment. For instance, in the above when rule 1 is applied the skin compartment contains $a^{c(q)+2}$. If the elements of U were not at least 3 units apart from each other there could be a rule using the number of occurrences of a's together with other a's present in the skin compartment. The application of this rule could let Π not to simulate M. As we ask the elements of U to be at least 3 units apart from each other, then this cannot happen.

When only $f(s_f)$ occurrences of a are present in the skin compartment, then the application of rule 13 halts the computation.

The simulation of M is faithful, that is Π cannot simulate sequences of instructions that cannot be performed by M. This means that if Π starts with a configuration in which $a^{f(s_1)}$ is present in the skin compartment and $a^{2val(\gamma_1)}$ is present in compartment 2, then Π halts with no a's in any of its compartments if and only if M would reach the final state with all registers empty if it started from the configuration $(s_1, val(\gamma_1), 0, \ldots, 0)$.

It is important to notice that the presence of catalysts is essential. If they were not present, then, for instance, rules 17 and 20 would be equal. Moreover, the fact that different rules in the same compartment share the same catalyst (as, for instance, rules 20 and 21) does not allow these rules to be used at the same time. □

Considering Theorem 5 and Proposition 1, we have:

Corollary 3.

$\mathbb{N}_1\mathsf{aO}_1\mathsf{P}_7(pmcat_{15}) = \mathbb{N}_1\mathsf{RE}$;
$\mathbb{N}\mathsf{aO}_1\mathsf{P}_{7,-c}(pmcat_{15}) = \mathbb{N}\,\mathsf{RE}$;
$\mathbb{N}_1\mathsf{aO}_1\mathsf{P}_5(pmcat_{11}) = \mathbb{N}_1\mathsf{RE}$ *if a specific input format is used;*
$\mathbb{N}\mathsf{aO}_1\mathsf{P}_{5,-c}(pmcat_{11}) = \mathbb{N}\,\mathsf{RE}$ *if a specific input format is used;*
$\mathbb{N}\mathsf{aO}_1\mathsf{P}_{3,-c}(pmcat_7) \supseteq \mathbb{N}\,\mathsf{REG}$.

5 Final Remarks

Theorem 3 partially answers suggestion for research 5.3 in [6] and it solves a problem stated in [11].

The suggestion for research in [6] proposed to understand why maximal parallelism was needed for a model of P systems to have a computational power equivalent to the one of partially blind register machines. Here we proved that increasing the number of compartments, these devices can simulate program machines.

The problem in [11] asked to investigate weather P systems with symport/antiport using only one symbol and operating under maximal parallelism are universal. Theorem 3 solves this problem.

We did not succeed in determining the power of the P systems considered in Theorems 3 and Theorem 5 when they operate in a sequential (asynchronous) mode.

It is not known if M1SV defines the family of unique-sum sets having minimum sun in function of their number of elements.

Acknowledgements

We thankfully acknowledge the feedback Oscar Ibarra, György Vaszil, Rudy Freund, Artiom Alhazov and one anonymous referee provided to previous versions of the present paper.

References

1. Alhazov, A., Freund, R.: P systems with one membrane and symport/antiport rules of five symbols are computationally complete. In: Gutiérrez-Naranjo, M.A., Riscos-Núñez, A., Romero-Campero, F.R., Sburlan, D. (eds.) Proceedings of the Third Brainstorming week on Membrane Computing, Sevilla, Spain, January 31 - February 4, pp. 19–28 (2005), Fénix Editoria, Sevilla (2005),

2. Alhazov, A., Freund, R., Oswald, M.: Symbol/membrane complexity of P systems with symport/antiport. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 96–113. Springer, Heidelberg (2006)
3. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
4. Ja Barzin, M.: On a certain class of Turing machines (Minsky machines. Algebra i Logika 1(6), 42–51 (1963) (in Russian) MR 27 #2415.
5. Frisco, P.: On s-sum vectors. Technical report, Heriot-Watt University, HW-MACS-TR-0058 (2008), http://www.macs.hw.ac.uk:8080/techreps/build_table.jsp
6. Frisco, P.: Computing with Cells. Advances in Membrane Computing. Oxford University Press, Oxford (2009)
7. Frisco, P.: Conformon P systems and topology of information flow. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 30–53. Springer, Heidelberg (2010)
8. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theoretical Computer Science 7, 311–324 (1978)
9. Hopcroft, J.E., Ullman, D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
10. Ibarra, O.H.: On membrane hierarchy in P systems. Theoretical Computer Science 334, 115–129 (2005)
11. Ibarra, O.H., Woodworth, S.: On symport/antiport P systems with small number of objects. International Journal of Computer Mathematics 83(7), 613–629 (2006)
12. Minsky, M.L.: Computation: Finite and Infinite Machines. In: Automatic computation, Prentice-Hall, Englewood Cliffs (1967)
13. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Computing 20(3), 295–306 (2002)
14. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 1(61), 108–143 (2000)
15. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
16. Păun, G., Pazos, J., Pérez-Jiménez, M.J., Rodriguez-Paton, A.: Symport/antiport P systems with three objects are universal. Fundamenta Informaticae 64, 1–4 (2005)
17. Qi, Z., You, J., Mao, H.: P systems and Petri nets. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 286–303. Springer, Heidelberg (2004)
18. The P Systems Webpage, http://ppage.psystems.eu/

An Integrated Approach to P Systems Formal Verification

Marian Gheorghe[1,2], Florentin Ipate[2],
Raluca Lefticaru[2], and Ciprian Dragomir[1]

[1] Department of Computer Science, University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk
[2] Department of Computer Science, University of Pitesti
Str Targu din Vale 1, 110040 Pitesti, Romania
florentin.ipate@ifsoft.ro, raluca.lefticaru@gmail.com

Abstract. This paper presents a method to formally verify P system specifications by first identifying invariants and then checking them, using the NuSMV model checker, against a Kripke structure representation. The method is applied to a basic class of P systems with transformation and communication rules using either maximal parallelism or asynchronous rewriting strategy and for a special variant of P systems with electrical charges, but without active membranes.

1 Introduction

P systems, introduced in [19], represent a new computational model inspired by the structure and functioning of the living cell. In the last ten years there have been various investigations related to this computational paradigm, ranging from computability and complexity for different variants of these systems [20] to various applications and connections with other computational models [4]. In the last period the research on various programming approaches to P systems ([8], [22]) and formal semantics ([3], [1], [15]), or with respect to decidability of some model checking properties [7], has created the basis for investigating different aspects related to the formal verification and testing of these systems.

The testing has been investigated for some covering criteria ([11], [13]) and certain formal based approaches [14].

Formal verification has been studied for different variants of P systems by using rewriting logic and the Maude tool [1] or, for stochastic systems [2], PRISM and associated probabilistic temporal logic [12].

In this paper we consider an integrated method for formally verifying P systems. A method for identifying invariants in a formal specification and a tool, NuSMV, that checks such properties against a Kripke structure representation is presented. This method is applied for a basic class of P systems, with transformation and communication rules using either maximal parallelism or asynchronous rewriting strategy and for a special variant of P systems with electrical charges, but without active membranes. The invariants are extracted from traces of simulations of P systems represented in P-Lingua.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 226–239, 2010.

2 Basic Definitions and Preliminary Relationships

2.1 P Systems

A basic cell-like P system is defined as a hierarchical arrangement of compartments delimited by membranes. Each compartment may contain a finite multiset of objects and a finite set of rules, as well as a finite set of other compartments. The rules perform transformation and communication operations. The class of such models will be called *transformation-communication P systems*.

Definition 1. *A* P system *is a tuple* $\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n)$*, where* V *is a finite set, called* alphabet*;* μ *defines the membrane structure, i.e., the hierarchical arrangement of* n *compartments called* regions *delimited by* membranes*; the membranes and regions are identified by integers 1 to* n*;* w_i*,* $1 \leq i \leq n$*, represents the initial multiset occurring in region* i*;* R_i*,* $1 \leq i \leq n$*, denotes the set of processing rules applied in region* i*.*

The membrane structure, μ, is denoted by a string of left and right brackets ([, and]), each with the label of the membrane it points to and describing the position of this membrane in the hierarchy. The rules in each region have the form $u \rightarrow (a_1, t_1)...(a_m, t_m)$, where u is a multiset of symbols from V, $a_i \in V$, $t_i \in \{in, out, here\}$, $1 \leq i \leq m$. When such a rule is applied to a multiset u in the current compartment, u is replaced by the symbols a_i. The symbols a_i with $t_i = here$, remain in the compartment; if $t_i = out$, then they are sent to the outer compartment or outside the system when the current compartment is the external one; when $t_i = in$, the symbols are sent into one of the compartments contained in the current one, arbitrarily chosen. In the following definitions and examples all the symbols $(a_i, here)$ are used as a_i. The rules are applied in maximally parallel mode which means that they are used in all the compartments at the same time and in each compartment all the objects to which a rule *can* be applied it *must* be the subject of a rule application [19].

A *configuration* of the P system Π, is a tuple $c = (u_1, ..., u_n)$, where $u_i \in V^*$, is the multiset associated with compartment i, $1 \leq i \leq n$. A *computation* from a configuration c_1 to c_2 using the maximal parallelism mode is denoted by $c_1 \Longrightarrow c_2$.

A configuration, $c = (u_1, ..., u_n)$, is a *terminal configuration* if there is no compartment i such that u_i can be further developed.

Another variant of P systems considered in this paper will be the *P systems with electrical charges*. This is a simplification of the usual variant occurring in the literature [21]. Each compartment has a specific electrical charge $(+, -, 0)$ which can be changed by a communication rule. The set of electrical charges is denoted by H. The set of rules contains the following types of rules:

- $[u \rightarrow v]_b^h$;
- $u[]_b^{h_1} \rightarrow [v]_b^{h_2}$;
- $[u]_b^{h_1} \rightarrow v[]_b^{h_2}$;

where b indicates a compartment and $h, h_1, h_2 \in H$. The rules are applied in the normal way; for more details see [21].

The maximal parallelism can be replaced by other execution strategies. One of them, called *asynchronous execution mode*, implies that at each step at least one rule is executed.

In the sequel we will consider transformation-communication P systems using maximal parallelism or with asynchronous behaviour or P systems with electrical charges and maximal parallelism.

2.2 Kripke Structures

Definition 2. *A Kripke structure over a set of atomic propositions AP is a four tuple $M = (S, H, I, L)$, where S is a finite set of states; $I \subseteq S$ is a set of initial states; $H \subseteq S \times S$ is a transition relation that must be left-total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $(s, s') \in H$; $L : S \longrightarrow 2^{AP}$ is an interpretation function, that labels each state with the set of atomic propositions true in that state.*

In general, the Kripke structure representation of a system consists of sets of values associated to the system variables. Assuming that $var_1, \ldots, var_n$ are the system variables and Val_i the set of values for var_i, with val_i a value from Val_i, $1 \leq i \leq n$, we can introduce the states of the system as

$$S = \{(val_1, \ldots, val_n) \mid val_1 \in Val_1, \ldots, val_n \in Val_n\}.$$

The set of atomic predicates are given by $AP = \{(var_i = val_i) \mid 1 \leq i \leq n, val_i \in Val_i\}$. Naturally, L will map each state (given by the values of system variables) onto the corresponding set of atomic propositions.

Additionally, a halt (sink) state is needed when H is not left-total and an extra atomic proposition, that indicates that the system has reached this state, is added to AP.

2.3 Linear Temporal Logic (LTL)

The most widely used query languages in model checking are based on *Linear Temporal Logic* (LTL) [17,18] and the branching time logic CTL (*Computation Tree Logic*) [5]. A superset of these logics is CTL* [9], which combines both linear-time and branching-time operators. A state formula in CTL* may be obtained from a path formula by prefixing it with a path quantifier, either **A** (for every path) or an **E** (there exists a path).

In LTL the only path quantifier allowed is **A**, i.e. we can describe only one path property per formula and the only state subformulas permitted are atomic propositions. More precisely, LTL formulas satisfy the following rules [6]:

- If $p \in AP$, then p is a path formula
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$ and $f\mathbf{R}g$ are path formulas, where:

- The **X** operator ("neXt time") requires that a property holds in the next state of the path.
- The **F** operator ("eventually" or "in the future") is used to assert that a property will hold at some state on the path.
- **G**f ("always" or "globally") specifies that a property, f, holds at every state on the path.
- f**U**g operator (**U** means "until") holds if there is a state on the path where g holds, and at every preceding state on the path, f holds. This operator requires that f has to hold *at least* until g, which holds at the current or a future position.
- **R** ("release") is the logical dual of the **U** operator. It requires that the second property holds along the path up to and including the first state where the first property holds. However, the first property is not required to hold eventually: if f never becomes true, g must remain true forever.

2.4 Transformation-Communication P Systems and Kripke Structure

In this section, following the presentation from [14], it is shown how a P system operating in a maximal parallel manner can be transformed into a Kripke structure. Then this will be adapted for other types of P systems. We only consider 1-membrane P systems in order to simplify the presentation. The approach presented below can be generalised for membrane systems with arbitrary number of compartments.

Consider a 1-membrane P system $\Pi = (V, \mu, w, R)$, where $R = \{r_1, \ldots, r_m\}$; each rule r_i, $1 \leq i \leq m$, is of the form $u_i \longrightarrow v_i$, where u_i and v_i are multisets over the alphabet V. In the sequel, we treat the multisets as vectors of non-negative integers. If k denotes the number of symbols in V and u a multiset of V, then we will write $u \in \mathbf{N}^k$.

The Kripke structure associated to Π utilises two predicates, $MaxPar$ and $Apply$ (similar to [7]):

$$MaxPar(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m), u \in \mathbf{N}^k, n_1, \ldots, n_m \in \mathbf{N},$$

$$Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m), u, v \in \mathbf{N}^k, n_1, \ldots, n_m \in \mathbf{N}.$$

The first predicate shows that a computation from the configuration u in maximally parallel mode is obtained by applying the rules $r_1 : u_1 \longrightarrow v_1, \ldots, r_m : u_m \longrightarrow v_m$, $n_1, \ldots, n_m$ times, respectively, to u; in particular, $MaxPar(u, u_1, v_1, 0, \ldots, u_m, v_m, 0)$ signifies that no rule can be applied and so u is a terminal configuration.

The predicate $Apply$ denotes that v is obtained from u by applying rules $r_1, \ldots, r_m$, $n_1, \ldots, n_m$ times, respectively.

In order to keep the number of configurations finite, we will assume that, for each configuration $u = (u^{(1)}, ..., u^{(k)})$, each component, $u^{(i)}$, $1 \leq i \leq k$, cannot exceed an established upper bound, denoted Max and, in each computation, each rule can only be applied for at most a given number of times, denoted Sup.

We denote $u \leq Max$ whenever $u^{(i)} \leq Max$ for every $1 \leq i \leq k$ and similarly $(n_1, \ldots, n_m) \leq Sup$ if $n_i \leq Sup$ for every $1 \leq i \leq m$; $\mathbf{N}^k_{Max} = \{u \in \mathbf{N}^k \mid u \leq Max\}$, $\mathbf{N}^m_{Sup} = \{(n_1, \ldots, n_m) \in \mathbf{N}^m \mid (n_1, \ldots, n_m) \leq Sup\}$. Analogously to [7], the system is assumed to crash whenever $u \leq Max$ or $(n_1, \ldots, n_m) \leq Sup$ does not hold (this is different from the normal termination, which occurs when $u \leq Max$, $(n_1, \ldots, n_m) \leq Sup$ and no rule can be applied). Under these conditions, the 1-membrane P system Π can be described by a Kripke structure $M_\Pi = (S, H, I, L)$ with $S = \mathbf{N}^k_{Max} \cup \{Halt, Crash\}$ with $Halt, Crash \notin \mathbf{N}^k_{Max}$, $Halt \neq Crash$; $I = w$ and H defined by:

- $(u, v) \in H$, $u, v \in \mathbf{N}^k_{Max}$, if $\exists (n_1, \ldots, n_m) \in \mathbf{N}^m_{Sup} \setminus \{(0, \ldots, 0)\} \cdot$ $MaxPar(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m) \wedge$ $Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$;
- $(u, Halt) \in H$, $u \in \mathbf{N}^k_{Max}$, if $MaxPar(u, u_1, v_1, 0, \ldots, u_m, v_m, 0)$;
- $(u, Crash) \in H$, $u \in \mathbf{N}^k_{Max}$, if $\exists (n_1, \ldots, n_m) \in \mathbf{N}^m, v \in \mathbf{N}^k \cdot$ $\neg((n_1, \ldots, n_m) \leq Sup \wedge v \leq Max) \wedge MaxPar(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$ $\wedge\ Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$;
- $(Halt, Halt) \in H$;
- $(Crash, Crash) \in H$.

It can be observed that the relation H is left-total. It is easy to show that for every $u, v \in \mathbf{N}^k_{Max}$, v is computed from u, in Π, if and only if $(u, v) \in H$, hence Π and M_Π show the same behaviour.

In the rest of our presentation we will consider a more compact form of the Kripke structure, M_Π, whereby all the states $\mathbf{N}^k_{Max}$ will be replaced by one *Running* state. So, *Running* state will define a state of normal behaviour as opposed to the situation described by the other two states, $Halt$ and $Crash$. In the first case the system stops in normal circumstances whereas in the case of the $Crash$ state, the system fails by going beyond some initially set finite limits.

Note that, in this section, the Kripke structure representation of a P system is given for maximal parallelism. On the other hand, the associated Kripke structure of a P system can be similarly constructed for other execution modes as well (e.g. asynchronous rewriting strategy) as illustrated by the examples given in the next section.

3 Transforming P Systems to NuSMV Specifications

In this section it is shown how the P systems considered in this paper will be mapped into the NuSMV model checker by adequately codifying Kripke structures associated with them in accordance with the principles presented in [14] and the description made in Section 2.4. Simple examples will illustrate the presentation.

3.1 Transformation-Communication P Systems to NuSMV Specifications

The presentation will follow the general principles, introduced in [14], for translating such P systems into NuSMV. The presentation below will be illustrated by

the following example: $\Pi_1 = (V, []_1, w_1, R_1)$, where $V = \{a, b, c, d, x, y\}$, $w_1 = xy$, $R_1 = \{r_1 : x \to a, r_2 : y \to b, r_3 : a \to xc, r_4 : b \to ydd\}$. Please note that the system will not halt, but this is less significant in this context. A computation in Π_1 has the following steps

$$xy \Longrightarrow ab \Longrightarrow xcydd \Longrightarrow acbdd \Longrightarrow xccydddd \ldots \Longrightarrow xc^n yd^{2n} \Longrightarrow ac^n bd^{2n} \ldots$$

In the NuSMV codification, we will use for each symbol, a, $a \in V$, above, a variable with the same name to denote the number of occurrences of this symbol in compartment 1 in the current step. If more than a compartment is considered then this variable should be indexed by the compartment number. For each rule r_i, we will identify by `ni` the NuSMV variable that expresses the number of times r_i is applied in the current step in a maximally parallel manner - this is the value that appears in the $MaxPar$ predicate.

We will describe a computation step in NuSMV by a transition from the state *Running* to itself in the Kripke structure associated with Π_1, M_{Π_1}. The maximal parallelism is expressed by the condition

```
x - next(n1) = 0 & y - next(n2) = 0 & a - next(n3) = 0 & b - next(n4) = 0
```

Additional conditions to characterise the *Running* state as well as equations to compute the values of the multisets in the next step are provided below.

```
state = running & next(state) = running &
x - next(n1) = 0 & y - next(n2) = 0 & a - next(n3) = 0 & b-next(n4) = 0 &
next(x) = x - next(n1) + next(n3) &
next(y) = y - next(n2) + next(n4) &
next(a) = a - next(n3) + next(n1) &
next(b) = b - next(n4) + next(n2) &
next(c) = c + next (n3) &
next(d) = d + 2 * next (n4) &
! (next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0) &
! (step >= MaxSteps | next(a) > Max | next(b) > Max | next(c) > Max |
   next(d) > Max | next(x) > Max | next(y) > Max | next(n1) > Sup |
   next(n2) > Sup | next(n3) > Sup | next(n4) > Sup )
```

The entire text providing details about the other transitions in the Kripke structure is available from the Appendix.

3.2 Asynchronous Transformation-Communication P Systems Mapped to NuSMV Specifications

We will illustrate the approach using the previous example, Π_1, but we will denote it by Π_2 as it runs in a different way. In the case of asynchronous behaviour the Kripke structure is slightly different. Although it is still possible to use the same states, the transitions will be different, as they reflect the asynchronous behaviour. The above mentioned NuSMV condition expressing maximal parallelism, becomes now

```
next(n1) + next(n2) + next(n3) + next(n4) > 0
```

showing that at least one rule is applied. The entire set of conditions for the *Running* state as well as equations defining the transition from this state to itself are listed below.

```
state = running & next(state) = running &
( next(n1) + next(n2) + next(n3) + next(n4) > 0)  &
( 0 <= next(n1) & next(n1) <= x &
  0 <= next(n2) & next(n2) <= y &
  0 <= next(n3) & next(n3) <= a &
  0 <= next(n4) & next(n4) <= b) &
next(a) = a - next(n3) + next(n1) &
next(b) = b - next(n4) + next(n2) &
next(c) = c + next(n3) &
next(d) = d + 2*next(n4) &
next(x) = x - next(n1) + next(n3) &
next(y) = y - next(n2) + next(n4) &
! (next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0  )&
! (step >= MaxSteps | next(a) > Max | next(b) > Max | next(c) > Max |
   next(d) > Max | next(x) > Max | next(y) > Max | next(n1) > Sup |
   next(n2) > Sup | next(n3) > Sup | next(n4) > Sup )
```

3.3 P Systems with Electrical Charges Mapped to NuSMV Specifications

In the case of electrical charges the above mentioned Kripke structures associated with P systems need to be extended to cope with additional conditions required in this case.

We will consider the following example of a P system with electrical charges and two compartments. $\Pi_3 = (V_3, [[]_2]_1, w_1, w_2, R)$, where $V_3 = \{a, b, c, d, x, y\}$, $w_1 = xy$, $w_2 = \lambda$, $R = \{r_1 : x[]_2^0 \rightarrow [a]_2^+, r_2 : y[]_2^0 \rightarrow [b]_2^+, r_3 : [a \rightarrow xc]_2^+, r_4 : [b \rightarrow ydd]_2^+, r_5 : [x]_2^+ \rightarrow []_2^0 x, r_6 : [y]_2^+ \rightarrow []_2^0 y\}$. The maximal parallelism strategy will be applied in running the system.

The Kripke structure associated with this example will codify the behaviour of the two-compartment P system by using a mapping of the symbols of the alphabet into compartments.

The main change to the Kripke structure represented in NuSMV for transformation-communication P systems consists in adding a new set of conditions, that represent the restrictions imposed by electrical charges associated with compartments. These constraints allow only for some rules to be applied. Two excerpts of NuSMV text for the P system Π_3 are listed below. The first one shows that the second membrane polarisation will become positive after applying rules like $r_1, \ldots r_4$ and neutral in case at least one of the rules r_5, r_6 is applied. Otherwise, if no rule is applied, the polarisation does not change. The second code excerpt shows how the restrictions are applied, e.g. each rule can be applied if the membrane charge is appropriate.

```
next(charge_2) := case
    next(n1) > 0 | next(n2) > 0 | next(n3) > 0 | next(n4) > 0  : 1;
    next(n5) > 0 | next(n6) > 0 : 0;
```

```
    1 : charge_2;
esac;

state = running & next(state) = running &
(( charge_2 = 0 & (x_1 > 0 | y_1 > 0 ) &
     x_1 - next(n1) = 0 & y_1 - next(n2) = 0 &
     next(n3) = 0 & next(n4) = 0 & next(n5) = 0 & next(n6) = 0 ) |
 ( charge_2 = 1 & (a_2 > 0 | b_2 > 0 ) &
     a_2 - next(n3) = 0 & b_2 - next(n4) = 0 &
     next(n1) = 0 & next(n2) = 0 & next(n5) = 0 & next(n6) = 0) |
 ( charge_2 = 1 & (x_2 > 0 | y_2 > 0 ) &
     x_2 - next(n5) = 0 & y_2 - next(n6) = 0 &
     next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0) ) &
next(x_1) = x_1 - next(n1) + next(n5) &
next(y_1) = y_1 - next(n2) + next(n6) &
next(a_2) = a_2 - next(n3) + next(n1) &
next(b_2) = b_2 - next(n4) + next(n2) &
next(c_2) = c_2 + next(n3) &
next(d_2) = d_2 + 2 * next(n4) &
next(x_2) = x_2 - next(n5) + next(n3) &
next(y_2) = y_2 - next(n6) + next(n4) &
! ( next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0 &
    next(n5) = 0 & next(n6) = 0 ) &
! ( step >= MaxSteps | next(x_1) > Max | next(y_1) > Max |
    next(a_2) > Max | next(b_2) > Max | next(c_2) > Max |
    next(d_2) > Max | next(x_2) > Max | next(y_2) > Max |
    next(n1) > Sup | next(n2) > Sup | next(n3) > Sup |
    next(n4) > Sup )
```

4 Formal Verification Using NuSMV

In this section we will show how a P system mapped into NuSMV is verified for certain properties, by using model checking techniques. The properties that will be checked are first identified by Daikon, a tool which dynamically detects program invariants based on execution traces. Following the strategy exposed in [2], we synthesise the traces from the P-Lingua environment's execution data[8], and then run Daikon to generate an extended list of invariants, which help us formulate the LTL properties. The Daikon tool is even able to detect some mathematical relationships between various variables of the system, based on complex mathematical functions, not all of them expressible in NuSMV. Both, the translation from P-Lingua specification to NuSMV specification and the P-Lingua traces conversion to Daikon inputs are obtained in an automatic way.

We will refer to the three examples presented above and to a nondeterministic variant of the predator-prey problem [10] in order to illustrate what kind of properties we can check. There are various ways to classify the properties we aim to verify. In certain areas there have been identified specific types of queries

categorised as patterns [16]. We will refer to some of these in the presentation below. We will present those properties as they have been captured by the Daikon tool and as LTL expressions.

We have initially looked at those properties that state the main invariants of the system. These invariants represent one of the main sets of patterns in [16]. It is obvious that we have been after properties like "two times the number of c's equals the number of d's" in the examples given by Π_1 and Π_3. Indeed, these properties have been identified by the Daikon tool as

```
2 * c - d == 0
```

or

```
2 * orig(c) - orig(d) == 0
```

where `orig(c)` means `c` in the previous step.

This property is then checked with the LTL query `G ( 2 * c - d = 0)`. It is easy to observe that this invariant does not hold for the P system Π_2, which works in an asynchronous way. Indeed this is neither returned by Daikon nor verified by NuSMV. This is a good example of a property that returns a test sequence for our system.

Other properties that are extracted by Daikon from the traces generated by P-Lingua describe some expected properties of a correct model. For instance in all these examples we expect that the number of occurrences of each of the variables `a, b, x, y` is either 0 or 1. This is present in the list of properties identified by Daikon and the NuSMV model checker shows this is true (in Daikon they occur as `a one of { 0, 1 }` or `b is boolean` and in NuSMV are expressed as LTL expressions `G ( 0 <= a & a <= 1 )` or `G ( 0 <= b & b <= 1 )`).

Some properties extracted by Daikon reveal relationships between elements of the multiset across development, sometimes involving different steps. These are not always obvious and can be utilised to generate some further more complex conditions. For example for the first P system, Π_1, it is identified the property `2 * c - 2 * orig(a) == orig(d)` which links `c` with `a, d` occurring in the previous step. This property holds for this example as it is shown by NuSMV.

Daikon was able to identify simple forms of consequence patterns [16], when the two states appear one after the other. For the Π_1 P system a relationship between consecutive occurrences of c is stated as `(c == 0) ==> (orig(c) == 0)`, which is true as it is shown by the NuSMV formula `G((c=0)->(c_old=0))`.

The nondeterministic variant of the predator-prey problem can be defined by the following P system, $\Pi_{PP} = (V, [\,]_1, w_1, R_1)$, where $V = \{a, x, y, b\}$, $R_1 = \{ax \to xx, xy \to yy, y \to b\}$ and w_1 is the initial multiset. We considered simulations for $w_1 = a^{100}x^{100}y^{10}$ and simulations and verifications using NuSMV for $w_1 = a^{10}x^{10}y^5$. This system simulates the interplay between preys, x's, and predators, y's, in an environment with a fixed amount of resources, a's. Preys breed when resources are available and are eaten by predators which also die – the last rule. Various simulations and analysis made with Daikon reveal consistently some invariants of this problem; of them only `(b == 0) ==> (orig(b) == 0)`

and `(orig(a) == 0) ==> (a == 0)` are validated by NuSMV (`G ( (step > 1 & b = 0) -> b_old = 0)` and `G ( (step > 1 & a_old = 0) -> a = 0)`, respectively). These show that if at a moment in time the number of death predators is 0 then this is true for all the previous steps and if the resources are exhausted they will remain the same. Other potential invariants, like `(a <= x)` or `(x > y)` or `(b < x)`, are not true and NuSMV confirms this. They can be true only for some executions, but not in general, irrespective of the initial values.

A comprehensive list of Daikon invariants and associated LTL specifications for the above examples has been collected and is available from EvoMT website `http://fmi.upit.ro/evomt/psys/psys_daikon.html`.

5 Conclusions

This paper has investigated a methodology to verify P systems specifications, by first identifying these properties as invariants produced by the Daikon tool and then formally checking whether these are true, by using NuSMV. The benefits of this methods have been identified and assessed through some case studies.

The methods suffers from the well known scalability issues most of the model checking based approaches exhibit.

In our future research we aim to overcome some of the limitations of the above presented approach, by better codifications of the systems and better formulated properties to be checked. We also intend to link the formal verification with testing and to expand it to other classes of P systems, including stochastic P systems which start to be increasingly used in various applications.

Acknowledgements

This research of MG, FI and RL was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 643/2008. The authors would like to thank the anonymous reviewers for their useful comments and for suggesting the use of the prey-predator example to illustrate the interplay between various tools employed.

References

1. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theor. Comput. Sci. 373(3), 163–181 (2007)
2. Bernardini, F., Gheorghe, M., Romero-Campero, F.J., Walkinshaw, N.: A hybrid approach to modeling biological systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 138–159. Springer, Heidelberg (2007)
3. Ciobanu, G.: Semantics of P systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of Membrane Computing, ch. 16, pp. 413–436. Oxford University Press, Oxford (2010)
4. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): Applications of Membrane Computing. Natural Computing Series. Springer, Heidelberg (2006)

5. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
6. Clarke Jr, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (1999)
7. Dang, Z., Ibarra, O.H., Li, C., Xie, G.: On the decidability of model-checking for P systems. Journal of Automata, Languages and Combinatorics 11(3), 279–298 (2006)
8. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for P systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, ch. 17, pp. 437–454. Oxford University Press, Oxford (2010)
9. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC 1982, pp. 169–180. ACM Press, New York (1982)
10. Fontana, F., Manca, V.: Predator-prey dynamics in P systems ruled by metabolic algorithm. Biosystems 91, 545–557 (2008)
11. Gheorghe, M., Ipate, F.: On testing P systems. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 204–216. Springer, Heidelberg (2009)
12. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
13. Ipate, F., Gheorghe, M.: Testing non-deterministic stream X-machine models and P systems. Electronic Notes in Theoretical Computer Science 227, 113–126 (2009)
14. Ipate, F., Gheorghe, M., Lefticaru, R.: Test generation from P systems using model checking. Journal of Logic and Algebraic Programming 79(6), 350–362 (2010)
15. Kleijn, J., Koutny, M.: Petri nets and membrane computing. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) Handbook of membrane computing, ch. 15, pp. 389–412. Oxford University Press, Oxford (2010)
16. Monteiro, P.T., Ropers, D., Mateescu, R., Freitas, A.T., de Jong, H.: Temporal logic patterns for querying dynamic models of cellular interaction networks. Bioinformatics 24(16), 227–233 (2008)
17. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, FOCS 1977, pp. 46–57. IEEE, Los Alamitos (1977)
18. Pnueli, A.: The temporal semantics of concurrent programs. Theoretical Computer Science 13, 45–60 (1981)
19. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
20. Păun, G.: Membrane Computing: An Introduction. Springer, Heidelberg (2002)
21. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
22. Şerbănuţă, T., Ştefănescu, G., Roşu, G.: Defining and executing P systems with structured data in K. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 374–393. Springer, Heidelberg (2009)

Appendix

NuSMV Specification

Based on the P system specification, the tool developed by the authors generates a SMV file, that will be processed by the NuSMV model checker.

```
-- This is a 1-membrane P system working in a maximally parallel manner
-- The NuSMV file is automatically generated
-- The P system consists of:
-- Alphabet = [a, b, c, d, x, y]
-- Initial multiset = x, y
-- Rules:
-- r1 : x --> a
-- r2 : y --> b
-- r3 : a --> x, c
-- r4 : b --> y, d*2

MODULE main

VAR
   a : 0..15;
   b : 0..15;
   c : 0..15;
   d : 0..15;
   x : 0..15;
   y : 0..15;
   a_old : 0..15;
   b_old : 0..15;
   c_old : 0..15;
   d_old : 0..15;
   x_old : 0..15;
   y_old : 0..15;
   n1 : 0..15;
   n2 : 0..15;
   n3 : 0..15;
   n4 : 0..15;
   state : {running, halt, crash};
   step : 0..15;

DEFINE
   Max := 10;
   MaxSteps := 10;
   Sup := 10;

ASSIGN
   init(a) := 0;
   init(b) := 0;
   init(c) := 0;
   init(d) := 0;
```

```
   init(x) := 1;
   init(y) := 1;
   init(a_old) := 0;
   init(b_old) := 0;
   init(c_old) := 0;
   init(d_old) := 0;
   init(x_old) := 0;
   init(y_old) := 0;
   init(n1) := 0;
   init(n2) := 0;
   init(n3) := 0;
   init(n4) := 0;
   init(state) := running;
   init(step) := 0;

ASSIGN
   next(a_old) := a;
   next(b_old) := b;
   next(c_old) := c;
   next(d_old) := d;
   next(x_old) := x;
   next(y_old) := y;

   next(step) := case
      (step <= MaxSteps & state=running) :  step + 1;
      1 : step;
   esac;

TRANS

-- STATE = running
state = running & next(state) = running &
x - next(n1) = 0 & y - next(n2) = 0 & a - next(n3) = 0 & b-next(n4) = 0 &
next(a) = a - next(n3) + next(n1) &
next(b) = b - next(n4) + next(n2) &
next(c) = c + next(n3) &
next(d) = d + 2*next(n4) &
next(x) = x - next(n1) + next(n3) &
next(y) = y - next(n2) + next(n4) &
! (next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0  )&
! (step >= MaxSteps | next(a) > Max | next(b) > Max | next(c) > Max |
   next(d) > Max | next(x) > Max | next(y) > Max | next(n1) > Sup |
   next(n2) > Sup  | next(n3) > Sup | next(n4) > Sup ) |

state = running & next(state) = halt &
x - next(n1) = 0 & y - next(n2) = 0 & a - next(n3) = 0 & b-next(n4) = 0 &
next(a) = a - next(n3) + next(n1) &
next(b) = b - next(n4) + next(n2) &
next(c) = c + next(n3) &
next(d) = d + 2*next(n4) &
```

```
next(x) = x - next(n1) + next(n3) &
next(y) = y - next(n2) + next(n4) &
(next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0) &
! (step >= MaxSteps | next(a) > Max | next(b) > Max | next(c) > Max |
   next(d) > Max | next(x) > Max | next(y) > Max | next(n1) > Sup |
   next(n2) > Sup | next(n3) > Sup | next(n4) > Sup) |

state = running & next(state) = crash &
x - next(n1) = 0 & y - next(n2) = 0 & a - next(n3) = 0 & b-next(n4) = 0 &
next(a) = a - next(n3) + next(n1) &
next(b) = b - next(n4) + next(n2) &
next(c) = c + next(n3) &
next(d) = d + 2*next(n4) &
next(x) = x - next(n1) + next(n3) &
next(y) = y - next(n2) + next(n4) &
! (next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0) &
  (step >= MaxSteps | next(a) > Max | next(b) > Max | next(c) > Max |
   next(d) > Max | next(x) > Max | next(y) > Max | next(n1) > Sup |
   next(n2) > Sup | next(n3) > Sup | next(n4) > Sup) |

-- STATE = HALT
state = halt & next(state) = halt &
next(n1) = 0 & next(n2) = 0 & next(n3) = 0 & next(n4) = 0 &
next(a) = a & next(b) = b & next(c) = c & next(d) = d & next(x) = x &
 next(y) = y |

-- STATE = CRASH
state = crash & next(state) = crash &
next(n1) = n1 & next(n2) = n2 & next(n3) = n3 & next(n4) = n4 &
next(a) = a & next(b) = b & next(c) = c & next(d) = d & next(x) = x &
 next(y) = y

-- Simple LTL checks
LTLSPEC   G ( a = b )
LTLSPEC   G ( x = y )
LTLSPEC   G ( 0 <= a & a <= 1 )
LTLSPEC   G ( d mod 2 = 0)
LTLSPEC   G ( 0 <= x & x <= 1 )
LTLSPEC   G ( step > 1 & state = running -> a = x_old )
LTLSPEC   G ( 2 * c - d = 0)
```

Using the SRSim Software for Spatial and Rule-Based Modeling of Combinatorially Complex Biochemical Reaction Systems

Gerd Grünert and Peter Dittrich

Jena Center for Bioinformatics, Bio Systems Analysis Group,
Institute of Computer Science, Friedrich Schiller University Jena,
Ernst-Abbe-Platz 1-4, 07743 Jena, Germany
{Gerd.Gruenert,Peter.Dittrich}@uni-jena.de
http://www.biosys.uni-jena.de/

Abstract. The simulator software SRSim is presented here. It is constructed from the molecular dynamics simulator LAMMPS and a set of extensions for modeling rule-based reaction systems. The aim of this software is coping with reaction networks that are combinatorially complex as well as spatially inhomogeneous. On the one hand, there is a combinatorial explosion of necessary species and reactions that occurs when complex biomolecules are allowed to interact, e.g. by polymerization or phosphorilation processes. On the other hand, diffusion over longer distances in the cell as well as the geometric structures of sophisticated macromolecules can further influence the dynamic behavior of a system. Addressing the mentioned demands, the SRSim simulation system features a stochastic, particle based, spatial simulation of Brownian Dynamics in three dimensions of a rule-based reaction system.

1 Rule-Based Modeling in Space

Biological systems exhibit a high number of possible combinations between interacting proteins, frequently leading to huge molecules of interconnected protein compounds. Examples are the complexes assembled for RNA or DNA transcriptases, ATP synthases [26], mitotic checkpoint networks [18] or the death inducing signaling complex (DISC) [41]. Next to the resulting complex graphs of interacting proteins, there are post-translational modification to proteins, e.g. from phosphorilations. Basically, each modification pattern defines a new chemical species with its unique chemical behavior. Exemplary, this would result in a number of 2^{27} different species for the tumor suppressor protein p53 which comprises 27 phosphorilation sites [2]. Such a high number of species poses problems for the simulation with (partial) differential equations and stochastic algorithms. But it also becomes very hard to analyze and understand the "mechanics" of such a complex model. A possible remedy for stochastic simulations is discussed here [35].

Another possible solution to the problem of combinatorial explosion is proposed by the **domain-oriented approach** and rule-based modeling [27,15,16,8, 12]. In this scenario, **elementary molecules** consist of a set of **components** or

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 240–256, 2010.

domains which can be modified or bound to the components of other molecules. Components, sites, binding sites and domains are used synonymously in this article. The resulting complex species, formed from a connection of elementary molecules, are called **molecule graphs**.

Instead of using reactions between explicit species now, the reactions are replaced by implicit reaction rules, which are applicable to a certain subset of all possible complex molecular species. This subset is defined through an equivalence class given by a molecule graph pattern. Any complex molecule graph that contains the graph pattern as an isomorphic sub-pattern is included in the equivalence class. A pattern might for example be described as demonstrated in Figure 1.

There is a lot of software available for rule-based modeling, as for example Stochsim [27], BioNetGen [5], BIOCHAM [10], Moleculizer [23] or Pathway Logic Assistant [39] or Cellucidate. Nonetheless spatial aspects are mostly neglected in these approaches (except for Stochsim).

1.1 Spatial Aspects

Spatiotemporal heterogeneities in reaction systems are generally considered to be of high importance for many systems [3, 25, 19, 38]. This led to a variety of spatial simulation techniques, starting from deterministic, population-based, partial differential equations [28] towards stochastic simulation of single particles in discrete or continuous 3d space [40,9,21]. See [22,38] for an overview on spatial simulation systems.

Similar to and partially based on the approaches [4,34,43,20,11,37], we are using individual agents for each elementary molecule in the simulation. Synonymously with the agents, we are using the terms particles and elementary molecules here. The spatial simulation is carried out as an extension to the molecular dynamics simulator LAMMPS [31]. Each particle is represented by its position, its velocity, its species and the state of its components. The particles diffuse through the reactor and can push away other molecules if they come too close. When two molecules approach one another, a bimolecular reaction can happen between them, if they are fitting to a reaction pattern specified in the reaction rules and if the geometric constraints are met. If a reaction binds two elementary molecules together, bond forces are applied and their diffusion through the reactor is coupled, forming a complex molecule graph. Monomolecular reactions can be used to spontaneously break bonds in molecule graphs or to modify the component states of a molecule. More conventional reactions can also be used to completely exchange one molecule for another.

The inclusion of spatial aspects in the rule-based reaction systems results in an expressive simulation system for moderately sized systems. Up to 100000 simulated particles can still be run on a desktop system for about 10^6 timesteps in some hours of computing time. Though SRSim was designed for the simulation of biological systems, also designing and planning chemical computing experiments might be a good application. We simulated for example the formation of Sierpinski triangles [13] following the work of Winfree et al. and Rothemund et al. [42,33].

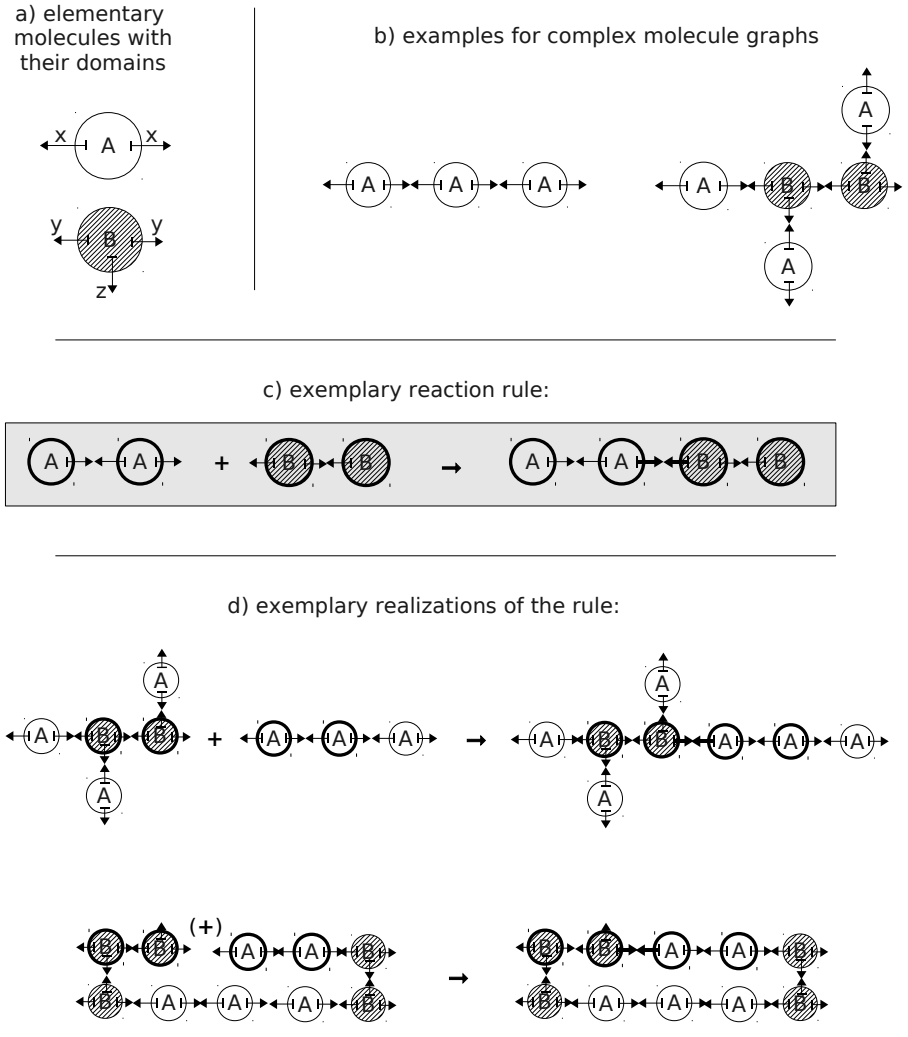


Fig. 1. Exemplary rule-based system, taken from [13]. Two elementary molecule types (A, B) with their sub domains (or **components**) are displayed (a). Each component can be bound to another component or be modified, e.g. denoting a phosphorylation or a conformational change. Site names need not be unique and hence a wide spectrum of possibilities for the system's specification is offered. Multiple elementary molecules can be connected at their components to form complex **molecule graphs** (b). Reaction rules, as the **binding reaction** (c), are specified by using patterns graphs (or **reactant patterns**) A reactant pattern fits to a molecule graph, if it is contained as a subgraph in the molecule graph. Note that some components are missing in the reactant pattern's definition, which are then ignored in the matching process. Panel (d) shows two different instances of the reaction rule. In the upper realization, two independent molecule graphs are connected. For the lower example on the other hand, both of the rules' reactant patterns are found in a single connected molecule graph.

Though the SRSim simulation system cannot directly be used to simulate P-Systems [29,30], there are some parallels to the membrane computing perspective. Similar to some P-System [24,32] we are using a stochastic simulation approach that is based on individual particles in 3d space, though with limited support for the constitution of membranes. That is, if we want to setup an implicit diffusion barrier in our system "SRSim", we can do this by adding forces to the reactor that confine certain molecules to defined subvolumes inside the reactor. Then, reactions can be used to transfer particles with specified rates through these pseudo-membranes. Nonetheless, that would be a rather awkward workaround to the ,,missing" membranes in the SRSim approach. On the other hand, similar to approaches like [11,36], the dynamic creation, modification and destruction of membranes can be reduced to the underlying macromolecular interactions by simulating lipid molecules that build the membranes.

Another similarity might be that both approaches, the rule-based and the membrane-computing systems, use further constraints on the underlying non-deterministic reaction system. While in membrane computing, there are dynamic membranes to separate different molecules against interactions, there are the geometry and the complex molecule-graph structure in the SRSim approach that allows or even favours one kind of reactions and that inhibits other types. For the inner workings of biological cells, both types of processes might be equally important. Maybe there could even be seen a hierarchy of first controlling geometries of interacting particles on the level of macromolecules and then constraining these interactions through the relationship between the compartments. From the computational point of view, both types of systems offer a high combinatorial complexity, leading to computational capacities as shown for P-Systems [30,36] and for self-assembly systems [1,33,7]. When intending to build computing systems from scratch, self-assembling macromolecules as well as structured membranes might both supply helpful building blocks.

Though that is not what we present in this paper, it might prove interesting to combine both types of constraints to a single system. This would also open the possibility to describe geometric relations not only between the particles, but also between the membranes. For the rule-based modelling community on the other hand, it would certainly be very handy to use the concept of dynamic membrane formation and decay. In the case of non-spatial simulation and static membranes, this was already done [14].

In the following sections, we try not to unfold the complete technical simulation process. Instead, the process of setting up and running a simple system will be demonstrated from installing the software to setting up and running the simulation. For the theory behind the spatial and rule-based simulation, please refer to [13].

2 Installing SRSim

Unfortunately the installation of SRSim is not yet fully automatized, so there are some uncomplicated steps to do. The following installation instructions are

addressed to x86 linux users, who are assumed to have Gnu Make and a C++ compiler installed. No tests were carried out using different hard- or software platforms, but as long as the required libraries are present, no architecture specific code is used.

2.1 Required Software

In the first place, make sure that the following libraries are present on your system, namely **Xerces-C++**[1], which is required for XML parsing. The other dependency is the "Message Passing Interface"[2] (**MPI**), a parallel computing standard used by LAMMPS. There are different MPI implementations available.

It is recommended but not necessary to install the software "Visual Molecular Dynamics"[3] (**VMD**) [17], which can be very helpful to visualize molecular trajectories calculated by SRSim.

2.2 Compiling SRSim

To build a SRSim executable, first the Rule System is compiled to a library that is later linked against the LAMMPS molecular dynamics simulator sources. After unpacking the SRSim distribution to a directory `X`, this should basically be done by invoking `make lmp_srsim` in the directory `X/source` of the SRSim distribution. This will create the library, build the tool `createGeo` and then compile LAMMPS with the additional modules necessary for SRSim.

If the MPI and Xerces libraries are not in the standard paths for include and library files, you have to modify the `-I` and `-L` paths in the makefiles `X/source/lammpsCompilation/Makefile` and `X/source/RuleSys/Makefile`.

After the successful compilation, two new executables, `X/source/lammpsCompilation/lmp_srsim` and `X/source/RuleSys/createGeo` can be found. You have to copy or link these files to a place in your system that is in your search path, as for example `/usr/local/bin` or `~/bin`. If you do not have a special directory for executables, you can just add the LAMMPS compilation directory directly to your search path by editing `~/.bashrc` and adding a line `export PATH=$PATH:X/source/lammpsCompilation:X/source/RuleSys`.

If the command `lmp_srsim` outputs the following line, you are done installing SRSim.

```
LAMMPS (7 Jul 2009)
```

Here, the 7th July 2009 is the LAMMPS version, that SRSim was built upon.

[1] Download from `http://xerces.apache.org/xerces-c/` or use the system's packet manager. Versions 2.7 and 2.8 seem to work fine.

[2] Download for example MPICH from `http://www.mcs.anl.gov/research/projects/mpich2/` or use your system's packet manager.

[3] Download from `http://www.ks.uiuc.edu/Research/vmd/`

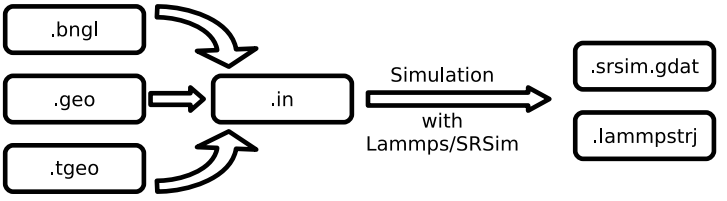


Fig. 2. Overview of the input / output file structure of SRSim

3 Using the Software

The molecular dynamics simulator LAMMPS is a script-driven command line program. Since SRSim is no autonomous tool, but an extension to LAMMPS, it is started in the same way as the Molecular Dynamics simulator: `lmp_srsim < input_script.in`. The LAMMPS input script `*.in` is then referencing three other input files and two or more output files as illustrated in Figure 2. The referenced input files are the `*.bngl` file, specifying the used rule-based reaction system, the `*.geo` file for the molecular geometry definition and finally the `*.tgeo` file for the template geometry definition. The output files will usually be a `*.srsim.gdat` file containing the concentrations of the observed species and a `*.lammpstrj` file with all the molecular coordinates, which can be used to visualize and to analyze the simulation run in detail.

To observe the results, gnuplot and VMD can be used. Type `vmd output.lammpstrj` for instance, to see a graphic representation of the reaction volume. Though VMD (See Section 2.1) was rather designed to display all-atom systems, it can be customized neatly and various helper scripts can be found on the Internet.

3.1 An Exemplary System

Let us assume we try to build a simple simulation with two elementary molecules. Species `A` will be a polymerizing molecule that requires energy provided by a molecule `B` to further polymerize. To allow a linear polymerization for molecules of type `A`, two opposing components `a` and `c` are introduced. A third component `b` will be used as the binding site for the type `B` molecules. The type `B` particles will only be able to bind to type `A` molecules, so only one binding site `b` is necessary. Nonetheless, to demonstrate the concept of modifications, we will also add another component `e` to the type `B` particles. Component `e` will be existing in two different energetic levels `~triP` and `~diP`. See figures 3 and 4 for an idea of what the system's behavior should be like, when simulated.

3.2 Definition of the Rule System

The rule-based reaction system is specified in the BioNetGen Language [16] (BNGL[4]). Basically it is not necessary to change a BNGL file to use it with

[4] See the BioNetGen Documentation. A BioNetGen tutorial can be found online at `http://bionetgen.org/index.php/BioNetGen_Tutorial` .

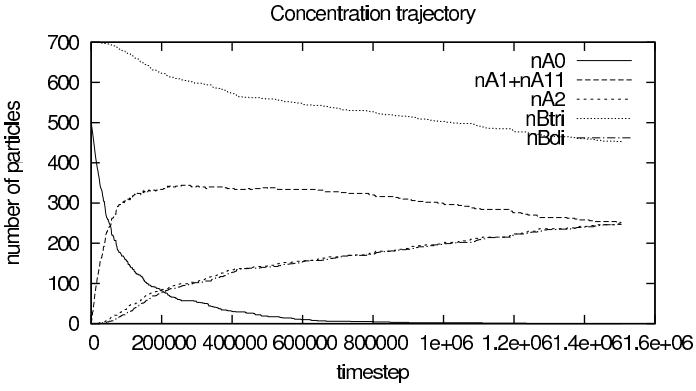


Fig. 3. Particle Quantity Trajectory. The displayed quantities will be defined and explained in the next Section about the rule system. The curve $nA1 + nA11$ denotes the number of particles "A" that have only either their site "a" or "c" bound. It can be observed, that this value increases rapidly in the beginning and falls down slowly then. This decrease is due to the larger complexes that are slowly forming in the later phase of the simulation.

SRSim. However, the commands `generate_network`, `simulate_ode`, `simulate_ssa` and `setConcentration` will be ignored. These commands' semantics can be replaced by using the LAMMPS input script instead, as we will explain later.

To define the reaction system, we first add the "parameters" and "species" blocks which allow the definitions of rate constants and the molecule types to appear in the simulation. Note that a molecular species is defined by its name followed by its components in brackets, optionally followed by a modification state. The constants `A0` and `B0` denote the initial quantity of particles of this type.

example1.bngl - part 1 of 3

```
begin parameters
  k1      1.5e-2
  k2      1.5e-2
  k3      5e-5

  A0      500
  B0      700
end parameters

begin species
  A(a,b,c)                  A0
  B(b,e~triP)               B0
end species
```

The next block defines which reactions are possible. A first rule is used to bind molecule `A` with a free `b` site to a molecule `B` with a free `b` site. The connections of

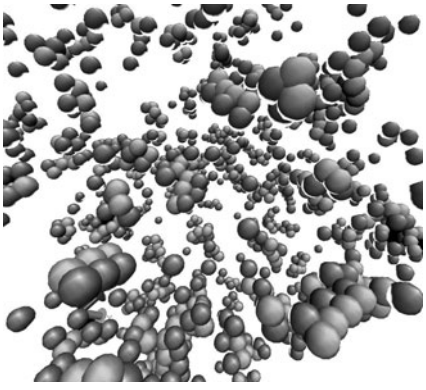


Fig. 4. A scene rendered from the reactor with VMD after 500000 timesteps. short multimerizations of two to four A-B dimers can be observed.

two molecules via their components is expressed through the exclamation mark, followed by a common identifier, `!1`. Since we did not mention any of the sites `a` or `c` of molecule `A`, they can be in any state, free or bound to any other complex molecule. To allow the polymerization of the `A` molecules, we need to define the second reaction rule. It should state, that a molecule `A` with a free binding site `a` can bind to another molecule `A'` with a free binding site `a'`. But only when one of them has bound an energy supplying molecule `B`. The third rule states, that a high-energy molecule `B` drops down into a lower energy state `~diP`, when the molecule `A` it belongs to has no free connection sites any more. Note that the binding symbol `a!+` marks a site that is bound to any other molecule that is not explicitly named.

example1.bngl - part 2 of 3

```
begin reaction rules
  1 A(b) + B(b)                      ->  A(b!1).B(b!1)                      k1

  2 A(c) + A(a,b!1).B(b!1,e~triP)  ->  A(c!2).A(a!2,b!1).B(b!1,e~triP)  k2

  3 A(a!+,c!+,b!1).B(b!1,e~triP)   ->  A(a!+,c!+,b!1).B(b!1,e~diP)      k3
end reaction rules
```

For the analysis of the reaction system, a fourth, optional block can be defined, listing patterns whose quantities should be output in the simulation. This might be for example the numbers of `A` molecules with no, one or two attached neighbors, or the numbers of `B` molecules in the high- or low energy state.

example1.bngl - part 3 of 3

```
begin observables
  Molecules    nA0    A(a,c)
  Molecules    nA1    A(a,c!+)
  Molecules    nA11   A(a!+,c)
```

```
  Molecules    nA2    A(a!+,c!+)
  Molecules    nBtri  B(e~triP)
  Molecules    nBdi   B(e~diP)
end observables
```

3.3 Molecule and Template Geometry Files

Now that the reaction system is defined, we have to specify the geometry for the elementary molecules `A` and `B` that we want to use in the `*.geo` geometry file. For each species, the mass and radius as well as the attributes for each component have to be defined. The orientations of the particles' binding sites are expressed as spherical coordinates through the angles phi, theta and a distance from the particles center. Phi can be imagined as the geographic longitude, while theta is similar to the geographic latitude. In contrast to geographic coordinates, `theta=0` specifies one pole, 90° is the equatorial plane and 180° is the other pole. For each elementary molecule type, a `molecule` section has to be defined. Within this, each site that is mentioned in the reaction system has to be represented by a `site` tag inside the molecule definition.

A general section in the beginning of the geometry file lists general property values for the simulation and values that should be used by default for all the particles. To specify certain values more individually, property tags can be included in the molecule and site blocks as well. See Table 1 for a list of property names and where they can be used.

A final section `DihedralAngles` can be used in the geometry definition to specify dihedral angles for certain bonds.

example1.geo - part 1 of 3

```
<?xml version="1.0"?>
<molecule-geometry-definition>
  <version value="1.01"/>

  <GeneralProperties>
     <property name="GPT_Devi_Dist"  value="0.2"/>
     <property name="GPT_Devi_Angle" value="40"/>
     <property name="GPT_Mol_Mass"   value="50"/>
     <property name="GPT_Mol_Rad"    value="1"/>
     <property name="GPT_Site_Dist"  value="1"/>

     <property name="GPT_Force_Repulsion"   value="1.5"/>
     <property name="GPT_Force_Bond"        value="1.5"/>
     <property name="GPT_Force_Angle"       value="1.5"/>
     <property name="GPT_Force_Dihedral"    value="1.5"/>
     <property name="GPT_Temperature"       value="300"/>

     <property name="GPT_Option_Dihedrals"  value="1"/>
     <property name="GPT_Option_Impropers"  value="0"/>
     <property name="GPT_Option_Rigid"      value="0"/>
  </GeneralProperties>
```

Table 1. Overview of the options that can be specified in the geometry file. The columns "glob" to "site" indicate, whether an option is (r)equired or (p)ossible on the (glob)al, the per-(mol)ecule, or the per-(site) level, respectively. Values that are defined in a more specialized context, e.g. for a special site overwrite the values that were specified in the more global context.

Property Name	glob	mol	site	function
GPT_Site_Theta	p	p	rp	spheric site coordinates
GPT_Site_Phi	p	p	rp	"
GPT_Site_Dist	p	p	rp	"
GPT_Site_Dihedral	p	-	-	not yet used
GPT_Mol_Mass	p	rp	-	molecular mass
GPT_Mol_Rad	p	rp	-	molecular radius
GPT_Devi_Dist	p	p	rp	max distance deviation for bond formation
GPT_Devi_Angle	p	p	rp	max angular deviation for bond formation
GPT_Diffusion	p	p	-	not yet used
GPT_Refractory	p	p	-	not yet used
GPT_Force_Repulsion	rp	-	-	factor for repulsive forces
GPT_Force_Bond	rp	-	-	factor for bond forces
GPT_Force_Angle	rp	-	-	factor for angular forces
GPT_Force_Dihedral	rp	-	-	factor for dihedral angles
GPT_Temperature	rp	-	-	not yet used
GPT_Option_Dihedrals	rp	-	-	use dihedrals 0/1
GPT_Option_Impropers	rp	-	-	not yet used
GPT_Option_Rigid	rp	-	-	use rigid bodies 0/1

```
  <molecule name="A">
     <site name="a"  phi="0" theta="0"   dist="1" />
     <site name="c"  phi="0" theta="180" dist="1" />
     <site name="b"  phi="0" theta="90"  dist="1">
        <property name="GPT_Devi_Angle" value="30"/>
     </site>
  </molecule>

  <molecule name="B">
     <property name="GPT_Mol_Mass" value="30"/>
     <site name="b"  phi="0" theta="0"   dist="1" />
     <site name="e"  phi="0" theta="180" dist="1" />
  </molecule>

  <DihedralAngles>
    <dihedral around="A(b,a!1).A(c!1,b)" angle="10" />
  </DihedralAngles>

</molecule-geometry-definition>
```

When complex molecule graphs are initially added to the simulation, the relative positions of the constituting elementary molecules have to be known. So they are specified manually or calculated in advance by an independent simulation step using the tool `createGeo` and stored in the template geometry `*.tgeo` files. In our case, this file looks very simple, since we are not adding complex molecules. More complex template definitions can be found in the other examples distributed with SRSim.

example1.tgeo

```
<?xml version="1.0"?>
<template-geometry-definition>

    <template id="0" name="A(a,b,c)">
    <mol id="0" x= "0" y="0" z= "0" />
    </template>

    <template id="1" name="B(b,e~triP)">
    <mol id="0" x= "0" y="0" z= "0" />
    </template>

</template-geometry-definition>
```

3.4 The LAMMPS Input Script

The LAMMPS input script is parsed line by line, each of which holds one command modifying the simulation system. Comments can be added using the `#` sign. Note that the order of the commands is important since the input script is parsed from top to bottom. There is a large number of possible commands that can be used to customize the simulation, so please refer to the LAMMPS documentation[5] for further details on the original LAMMPS commands. These commands can for example be used to define custom force terms or to create simulation outputs in different formats.

example1.in

```
##
#  Phase 1 - setup reactor
##

dimension     3
boundary      f f f       # use fixed boundary conditions
units         real        # timescale: fs,  distances: Angstrom
newton        on

atom_style    srsim example1.bngl example1.geo example1.tgeo 11111
##########    cmd     .bngl         .geo          .tgeo      random_seed
```

[5] The LAMMPS documentation comes together with LAMMPS' sources and can be accessed online at `http://lammps.sandia.gov/doc/Manual.html`.

```
lattice       none
region  Nucleus block -40  40  -40  40  -40  40 units box
######                      dimensions of the reaction volume

create_box    100           Nucleus
#             n_atom_types  Region_name

start_state_srsim   coeffs
start_state_srsim   atoms
#   set initial values e.g. bond forces etc.
#   and add molecules to the simulation

neighbor            5.0 bin
#                   size of neighbor-grouping bins

##
#  Phase 2 - setup forces
##

fix  1   all   langevin  300  300  160.0   12345
#        parameters:     Temp Temp Gamma^-1 random_seed

fix  2   all   nve
fix  3   all   wall/reflect xlo xhi ylo yhi zlo zhi

fix  4   all   srsim   1   45678  1.0  1.0  1.0  1.0  1.0  50
#  fix srsim syntax: fix id group srsim | nEvery  randomSeed
#                    preFactBindR preFactBreakR preFactExchangeR
#                    preFactModifyR_1 preFactModifyR_2 refractoryTime

##
#  Phase 3 - run
##

# Dumps:
thermo     5000   # write themodynamics information every 5k timesteps
timestep   1      # one timestep = 1  fs

dump            1 all atom 1000 example1.lammpstrj
                ################ trajectory output
dump_modify     1 scale yes
dump            2 all srsim 1000 example1.srsim.gdat
                ################# concentrations output

# first run phase for 500k ts
run         500000

# second run phase with higher time-resolution
dump_modify     1 every 10
run             5000
```

In the first phase, some basic parameters have to be set, as for example the units to be used, the size of the reaction volume, the maximum number of molecular species and the initial configuration of the simulation system. Note that the command `atom_style` uses a special atom style, specially designed for SRSim, which is followed by the names of the other input files and the random seed.

In the second phase, different "fixes" are selected to be applied to the simulation. These are computations which influence each molecule's data, for example their positions, velocities or binding states. The most basic fix, called `nve`, is the calculation applied to move each particle according to Newton's equations of motion in dependency of the applied forces. The fix `langevin` adds implicit solvent effects, resulting in Brownian movement of the particles. The second last parameter to the fix `langevin` is called damping factor (γ^{-1}). It depends directly on the diffusion coefficient D and the temperature T by $\gamma^{-1} = \frac{D}{k_B T}$, where k_B is the Boltzmann constant. Since fixed boundary conditions were chosen before, molecules moving out of the reaction volume would be lost, so the fix `wall/reflect` is applied. The last fix, `srsim` is the part of SRSim that checks for molecular collisions, analyzes which rules are applicable and finally executes them.

In the last phase, the types of output and the length of the simulation runs will be defined. The dump type `srsim` creates a plain text file in the same format as BioNetGen, to allow an easy comparison of the computed trajectories. Note that the intervals between two successive output data writes can be changed using the command `dump_modify`. If new molecules are to be added to the running simulation, the command `runmodif_srsim addMols` can be used, given the specified molecule-graph type was already listed in the reaction system definition.

3.5 The Tool "createGeo"

To simplify the creation of `.geo` and `.tgeo` files, the tool `createGeo` was added to the SRSim programs. It is used in the following syntax:

```
createGeo input.bngl input.geo input.tgeo
```

If either the `.geo` or the `.tgeo` file is not exiting, it will created. Molecule geometries are created with initial values of 1.0 for all distances and predefined angles for up to 6 sites. Template geometries are calculated by running short MD simulations to relax all bond distances and angles.

4 Concluding Remarks

In this manual, we have shown for a very simple system how to setup the SRSim simulation system. Not every possibility for configuration was mentioned, though. This is mostly due to the vast amount of options offered by the LAMMPS scripting language. Most of the molecular dynamics simulator's capabilities can still be used with SRSim, offering a great potential to describe a system's peculiarities. Another reason is, that SRSim is still under development and some

features are still changing or are not yet fully tested. There are other examples in the SRSim package which might convey more ideas on what is possible with the simulation system.

Features that are missing at the moment are reactions that can change the states of three or more components instantaneously. So at this stage of the development, it is not possible to have a binding reaction, that also changes the modification state of a component at the same time. Nonetheless a wide range of reaction systems can be expressed under these constraints and more features will be included in future releases of the software. Another aspect that is not covered by this paper on the SRSim software, is the analysis of the results. Though special systems will probably require customized methods for the analysis, a first idea of what happens in the reactor can mostly be obtained through molecular dynamics visualization tools. VMD [17] for example comes with an import filter for LAMMPS trajectories. It is also possible to extend VMD with python and tcl scripts for more specialized purposes.

SRSim can be interpreted as a P-System without explicit membranes. Reactions are constrained by spatial configurations and geometries instead of explicit membranes. So far, membranes can only be defined as static force fields or can emerge (e.g. like lipid layers formations [36]), which is computationally extremely demanding. Thus, we suggest to use explicit membranes like it is done in P-Systems, enriched by geometric information. In this approach, geometric properties like a form (e.g. sphere), a location, a size and a velocity are added to a membrane, so that it can have an effect on and can be affected by spatial heterogenities. For example, a reaction to make a molecule of species A leave a membrane x

$$[[..., A_i]_x ...]_y \longrightarrow [[...]_x A_i ...]_y$$

might require an appropriate particle A_i to be situated close to the membrane, before it can exit. Other constraints follow easily, e.g. mean transition times from one membrane into another compartment that is situated some distance away. Similar ideas were implemented in demonstrating software by Damien Pous[6] following the concepts of "mobile ambients" [6].

SRSim as well as LAMMPS are released under the GPL, so the sources can freely be downloaded[7] and modified. Especially the Rule System that handles the rule-based reaction system is independent from the molecular dynamics simulator and could be plugged into a different spatial or non-spatial realization of a rule-based simulation system.

Acknowledgements

The research was supported by the NEUNEU project (248992) sponsored by the European Community within FP7-ICT-2009-4 ICT-4-8.3 - FET Proactive 3: Bio-chemistry-based Information Technology (CHEM-IT) program.

[6] http://www-sop.inria.fr/mimosa/ambicobjs/

[7] www.biosystemsanalysis.de

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 266(5187), 1021–1024 (1994)
2. Arkin, A.P.: Synthetic cell biology. Curr. Opin. Biotechnol. 12(6), 638–644 (2001)
3. Berg, O.G., von Hippel, P.H.: Diffusion-controlled macromolecular interactions. Annu. Rev. Biophys. Biophys. Chem. 14(1), 131–158 (1985)
4. Berger, B., Shor, P.W., Tucker-Kellogg, L., King, J.: Local rule-based theory of virus shell assembly. Proc. Natl. Acad. Sci. U S A 91(16), 7732–7736 (1994)
5. Blinov, M.L., Faeder, J.R., Goldstein, B., Hlavacek, W.S.: Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics 20(17), 3289–3291 (2004)
6. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)
7. Conrad, M., Zauner, K.P.: Dna as a vehicle for the self-assembly model of computing. Biosystems 45(1), 59–66 (1998)
8. Conzelmann, H., Saez-Rodriguez, J., Sauter, T., Kholodenko, B.N., Gilles, E.D.: A domain-oriented approach to the reduction of combinatorial complexity in signal transduction networks. BMC Bioinformatics 7, 34 (2006)
9. Ermak, D.L., Mccammon, J.A.: Brownian dynamics with hydrodynamic interactions. J. Chem. Phys. 69(4), 1352–1360 (1978)
10. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. J. of biol. phys. and chem. 4, 64–73 (2004)
11. Fellermann, H., Rasmussen, S., Ziock, H., Solé, R.: Life cycle of a minimal protocell-a dissipative particle dynamics study. Artificial Life 13(4), 319–345 (2007)
12. Feret, J., Danos, V., Krivine, J., Harmer, R., Fontana, W.: Internal coarse-graining of molecular systems. Proc. Natl. Acad. Sci. U S A 106(16), 6453 (2009)
13. Gruenert, G., Ibrahim, B., Lenser, T., Lohel, M., Hinze, T., Dittrich, P.: Rule-based spatial modeling with diffusing, geometrically constrained molecules. BMC Bioinformatics 11(1), 307 (2010)
14. Harris, L.A., Hogg, J.S., Faeder, J.R.: Compartmental rule-based modeling of biochemical systems. In: Rossetti, M., Hill, R., Johansson, B., Dunkin, A., Ingalls, R. (eds.) Proceedings of the 2009 Winter Simulation Conference (2009)
15. Hlavacek, W.S., Faeder, J.R., Blinov, M.L., Perelson, A.S., Goldstein, B.: The complexity of complexes in signal transduction. Biotechnol. Bioeng. 84(7), 783–794 (2003)
16. Hlavacek, W.S., Faeder, J.R., Blinov, M.L., Posner, R.G., Hucka, M., Fontana, W.: Rules for modeling signal-transduction systems. Sci STKE 2006(344) re6 (2006)
17. Humphrey, W., Dalke, A., Schulten, K.: VMD – Visual Molecular Dynamics. Journal of Molecular Graphics 14, 33–38 (1996)
18. Ibrahim, B., Diekmann, S., Schmitt, E., Dittrich, P.: In-silico modeling of the mitotic spindle assembly checkpoint. PLoS ONE 3(2), e1555 (2008)
19. Karplus, M., McCammon, J.A.: Molecular dynamics simulations of biomolecules. Nat. Struct. Biol. 9(9), 646–652 (2002)

20. Kurth, W., Kniemeyer, O., Buck-Sorlin, G.: Relational growth grammars - a graph rewriting approach to dynamical systems with a dynamical structure. In: Banâtre, J.P., Fradet, P., Giavitto, J.L., Michel, O. (eds.) Unconventional Programming Paradigms, pp. 56–72. Springer, Berlin (2005)
21. Leach, A.: Molecular Modelling: Principles and Applications, 2nd edn. Prentice-Hall, Englewood Cliffs (2001)
22. Lemerle, C., Ventura, B.D., Serrano, L.: Space as the final frontier in stochastic simulations of biological systems. FEBS Lett. 579(8), 1789–1794 (2005)
23. Lok, L., Brent, R.: Automatic generation of cellular reaction networks with moleculizer 1.0. Nature Biotech. 23(1), 131–136 (2005)
24. Manca, V., Bianco, L., Fontana, F.: Evolution and oscillation in p systems: Applications to biological phenomena. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 63–84. Springer, Heidelberg (2005)
25. Minton, A.P.: The influence of macromolecular crowding and macromolecular confinement on biochemical reactions in physiological media. J. Biol. Chem. 276(14), 10577–10580 (2001)
26. Nakamoto, R.K., Scanlon, J.A.B., Al-Shawi, M.K.: The rotary mechanism of the atp synthase. Archives of Biochemistry and Biophysics 476(1), 43–50 (2008); special Issue: Transport ATPases
27. Novère, N.L., Shimizu, T.S.: Stochsim: modelling of stochastic biomolecular processes. Bioinformatics 17(6), 575–576 (2001)
28. Øksendal, B.: Stochastic Differential Equations: An Introduction with Applications. Springer, Heidelberg (2005)
29. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Berlin (1998)
30. Păun, G.: Introduction to Membrane Computing. In: Applications of Membrane Computing, pp. 1–42. Springer, Berlin (2006)
31. Plimpton, S.J.: Fast parallel algorithms for short-range molecular dynamics. J. Comp.Phys. 117, 1–19 (1995)
32. Romero-Campero, F., Pérez-Jiménez, M.: Modelling gene expression control using p systems: The lac operon, a case study. BioSystems 91(3), 438–457 (2008)
33. Rothemund, P., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology 2(12), e424 (2004)
34. Schwartz, R., Shor, P.W., Prevelige, P.E., Berger, B.: Local rules simulation of the kinetics of virus capsid self-assembly. Biophys. J. 75(6), 2626–2636 (1998)
35. Slepoy, A., Thompson, A., Plimpton, S.: A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. J. Chem. Phys. 128, 205101 (2008)
36. Smaldon, J., Krasnogor, N., Alexander, C., Gheorghe, M.: Liposome logic. In: Rothlauf, F. (ed.) GECCO 2009: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 161–168. ACM Press, New York (2009)
37. Sweeney, B., Zhang, T., Schwartz, R.: Exploring the Parameter Space of Complex Self-Assembly through Virus Capsid Models. Biophys. J. 94(3), 772 (2008)
38. Takahashi, K., Arjunan, S.N.V., Tomita, M.: Space in systems biology of signaling pathways–towards intracellular molecular crowding in silico. FEBS Lett. 579(8), 1783–1788 (2005)

39. Talcott, C., Dill, D.: The pathway logic assistant. In: Plotkin, G. (ed.) Proceedings of the Third International Workshop on Computational Methods in System Biology, pp. 228–239 (2005)
40. Verlet, L.: Computer "experiments" on classical fluids. I. Thermodynamical properties of lennard-jones molecules. Phys. Rev. 159(1), 98 (1967)
41. Weber, C.H., Vincenz, C.: A docking model of key components of the disc complex: death domain superfamily interactions redefined. FEBS Lett. 492(3), 171–176 (2001)
42. Winfree, E., Liu, F., Wenzler, L., Seeman, N.: Design and self-assembly of two-dimensional DNA crystals. Nature 394(6693), 539–544 (1998)
43. Zhang, T., Rohlfs, R., Schwartz, R.: Implementation of a discrete event simulator for biological self-assembly systems. In: Proceedings of the 37th Winter Simulation Conference, Orlando, FL, USA, December 4-7, 2005, pp. 2223–2231. ACM, New York (2005)

Depth-First Search with P Systems

Miguel A. Gutiérrez-Naranjo and Mario J. Pérez-Jiménez

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

Abstract. The usual way to find a solution for an NP complete problem in Membrane Computing is by brute force algorithms. These solutions work from a theoretical point of view but they are implementable only for small instances of the problem. In this paper we provide a family of P systems which brings techniques from Artificial Intelligence into Membrane Computing and apply them to solve the N-queens problem.

1 Introduction

Brute force algorithms have been widely used in the design of solutions for NP problems in Membrane Computing. Trading time against space allows us to solve NP problems in polynomial time with respect to the input data. The cost is the amount of resources, which grows exponentially. The usual idea of these brute force algorithms is to encode each feasible solution in one membrane. The number of candidates to solution is exponential in the input size, but the coding process can be done in polynomial time. Once generated all these candidates, each of them is tested in order to check whether it represents a solution to the problem or not. This checking stage is made simultaneously in all membranes by using massive parallelism. Next, the P system halts and sends a signal to the user with the output of the process. Such theoretical process works and different P system models have been explored by searching the limits between tractability and intractability [3]. In such way, several ingredients have been mixed and nowadays there exist many open problems in the area (see, e.g., [6]).

In spite of the great success in the design of theoretical solutions to NP problems, these solutions have an intrinsic drawback from a practical point of view. In all imaginable implementation, a membrane will have a *space* associated (maybe a piece of memory in a computer, a pipe in a lab or the volume of a bacterium) and brute force algorithms only will be able to implement little instances of such problems. As an illustration, if we consider an in vivo implementation where each feasible solution is encoded in an elementary membrane and such elementary membrane is *implemented* in a bacterium of mass similar to E. Coli ($\sim 7 \times 10^{-16}$ kg., see [9]), then, a brute force algorithm which solves an instance of an NP problem with input size 40 will need approximately the mass of the Earth for an implementation ($\sim 6 \times 10^{24}$ kg., *ibid.*).

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 257–264, 2010.

In this paper we explore the possibility of searching solutions to NP problems with Membrane Computing techniques, but taking ideas from Artificial Intelligence instead of using brute force algorithms. Of course, the worst case of any solution of an NP-problem needs an exponential amount of resources, but we are not always in the worst case. The contribution of using search strategies from Artificial Intelligence is that, on average, the number of resources for solving several instances of an NP problem decreases with respect to the number of resources used by brute force. As a case study, we present the N-queens problem (Section 2), previously studied in the framework of Membrane Computing in [2].

The paper is organized as follows: Next we present the N-queens problem and recall the algorithm presented in [2]. In Section 3, we give some brief notions of searching strategies in Artificial Intelligence and in Section 4, an implementation of depth-first search with P systems is shown. In Section 5, we present a family of P systems which solve the N-queens problem based on the cellular implementation. Finally, some conclusions and open research lines are presented.

2 The N-Queens Problem

Along this paper we will consider the N-queens problem as a case study. It is a generalization of a classic problem known as the 8-queens problem. It consists on putting N queens on an N×N chessboard in such way that none of them is able to capture any other using the standard movement of the queens in chess, i.e., at most one queen can be placed on each row, column and diagonal line.

In [2], a first solution to the N-queens problem in Membrane Computing was shown. For that aim, a family of deterministic P systems with active membranes was presented. In this family, the N-th element of the family solves the N-queens problem and the last configuration encodes *all* the solutions of the problem.

In order to solve the N-queens problem, a truth assignment that satisfies a formula in conjunctive normal form (CNF) is searched. This problem is exactly SAT, so the solution presented in [2] uses a modified solution for SAT from [7]. Some experiments were presented by running the P systems with an updated version of the P-lingua simulator [1]. The experiments were performed on a system with an Intel Core2 Quad CPU (a single processor with 4 cores at 2,83Ghz), 8GB of RAM and using a C++ simulator under the operating system Ubuntu Server 8.04.

According to the representation in [2], the 3-queens problem is expressed by a formula in CNF with 9 variables and 31 clauses. The *input multiset* has 65 elements and the P system has 3185 rules. Along the computation, $2^9 = 512$ elementary membranes need to be considered in parallel. Since the simulation was carried out on a uniprocessor system, these membranes were evaluated sequentially. It took 7 seconds to reach the halting configuration. It is the 117-th configuration and in this configuration one object `No` appears in the environment. As expected, this means that we cannot place three queens on a 3×3 chessboard satisfying the restriction of the problem.

In the 4-queens problem, we try to place four queens on a 4×4 chessboard. According to the representation, the problem can be expressed by a formula in CNF with 16 variables and 80 clauses. Along the computation, $2^{16} = 65536$ elementary membranes were considered in the same configuration and the P system has 13622 rules. The simulation takes 20583 seconds (> 5 hours) to reach the halting configuration. It is the 256-th configuration and in this configuration one object `Yes` appears in the environment. This configuration has two elementary membranes encoding the two solutions of the problem (see [2] for details).

According to this design, for the solution of the N-queens problem in a standard 8×8 chessboard $2^{64} = 18.446.744.073.709.551.616$ elementary membranes should be considered simultaneously. If we follow with the analogy from the Introduction, an *E. Coli implementation* of such P system will need approximately a metric ton of bacteria to solve the problem.

3 Searching Strategies

Searching has been deeply studied in Artificial Intelligence. In its basic form, a *state* is a description of the world and two states are linked by a *transition* which allows to reach a state from a previous one. In this way, a directed graph where the nodes are the states and the edges are the actions is considered. Given a starting state, a sequence of actions to one of the final states is searched.

In sequential algorithms, only one node is considered in each time unit and the order in which we explore new nodes determines the different searching strategies. In the usual framework, several possible unexplored nodes are reachable and we need to choose one of them in order to continue the search. In the best case, we have a heuristic which can help us to decide the best options among the candidates. Such heuristic represents, in a certain sense, how far the considered node is from a solution node and it captures our information about the nature of the problem. In many other situations we have no information about how far we are from a solution and we need to use a *blind strategy*. Since there is no information about the nature of the problem, blind strategies are based in the topology of the graph and the order in which new nodes are reached.

The two basic blind search strategies are depth-first search and breadth-first search. The main difference between them is that depth-first search follows a path to its completion before trying an alternative path. Some paths can be infinite, so this search may never succeed. It involves *backtracking*: One alternative is selected for each node and it backtracks to the next alternative when it has pursued all of the paths from the first choice. In the worst case, depth-first search will explore all of the nodes in the search tree. The complexity in time is linear on the maximum of the number of vertices and the number of edges and the complexity in space is quadratic. In breadth-first search the order in which nodes are explored depends on the number of arcs in the path. The algorithm always selects one of the paths with fewest arcs. In this case the complexity in time and in space is the same as for depth-first search.

4 Depth-First Search with P Systems

The idea of representing an instantaneous description of the world as a state and a transition from a state to the following one as an edge in the graph is so general that many real-life problems can be modeled as a problem of space of states. In this paper, a first approach to depth-first search with P systems is presented. The aim of this first approach is not to provide a minimalist approach. We are not looking for the minimum number of ingredients for implementing the depth-first search in P systems. In fact, we use four of the most powerful available ingredients: inhibitors, cooperation, priorities and dissolution. As we will remark in Section 6, it is an open question to weaken these conditions.

In an abstract way, a representation of a problem $P = (a, S, E, F)$ as a space of states consists of a set of states S and an initial state, $a \in S$; a set E of ordered pairs (x, y), called *transitions*, where x and y are states and y is reachable from x in one step and a set F of final states. Technically, a *cost* mapping is also needed, which assigns a cost to each transition (x, y), but we will consider a constant cost and we will omit it. Given a problem $P = (a, S, E, F)$, we will consider a P system $\Pi = (\Gamma, H, \mu, w_u, w_s, R_1, R_2, R_3, R_1 > R_2 > R_3)$ where

- The alphabet $\Gamma = S \cup \{p_x \,|\, x \in S\} \cup \{r_e \,|\, e \in E\}$
- The set of labels $H = \{u, s\}$
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- The initial multisets $w_u = \{a\}$ and $w_s = \emptyset$.
- The sets of rules R_1, R_2 and R_3 are associated with the membrane u:
  - $R_1 = \{[x]_u \to \lambda : x \in F\}$. For each final state we have a dissolution rule which dissolves the membrane u.
  - $R_2 = \{[x \neg p_y \to y\, r_{xy}]_u : (x, y) \in E\}$. For each transition (x, y), x produces $y\, r_{xy}$ if p_y does not occur in the membrane u, i.e., p_y acts as an inhibitor.
  - $R_3 = \{[y\, r_{xy} \to x\, p_y]_u : (x, y) \in E\}$. For each transition (x, y) we have a cooperative rule where the multiset $y\, r_{xy}$ is rewritten as $x\, p_y$ in the membrane u.
- An order among the rules is considered. Rules of R_1 have higher priority than the other rules and rules from R_2 have priority over rules from R_3.

In each configuration (but in the last one) there is one object from S in the configuration. It represents the current state in the searching process. For each state y, the object p_y is an inhibitor[1] which forbids to visit the state y. Finally, the occurrence of the object r_{xy} represents that the transition (x, y) belongs to the path from the initial state to the current one.

4.1 Example

Let us consider a representation of a problem as a space of states $P = (a, S, E, F)$ with $S = \{a, b, c, d, e, f, g\}$, a the initial state, the set of transitions $E = \{(a, b),$

[1] Notice that the object p_y is never removed. If the state y can be reached from different paths, then we should add new rules in order to prevent it.

$(a,c),(b,d),(b,e),(e,f),(c,g)\}$ and the set of final states $F=\{g\}$. Let Π be the P system associated with this space as described above. The initial configuration is $C_0 = [\,[a]_u\,]_s$. Two rules are applicable from the set R_2, $r_b \equiv [a\,\neg p_b \rightarrow b\,r_{ab}]_u$ and $r_c \equiv [a\,\neg p_c \rightarrow c\,r_{ac}]_u$. Let us suppose that non-deterministically r_b is chosen. Then $C_1 = [\,[b\,r_{ab}]_u\,]_s$ is obtained. From C_1, three rules are applicable $r_d \equiv [b\,\neg p_d \rightarrow d\,r_{bd}]_u \in R_2$, $r_e \equiv [b\,\neg p_e \rightarrow e\,r_{be}]_u \in R_2$ and $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_b]_u \in R_3$.

Since R_2 has priority over R_3, only r_d or r_e can be non-deterministically chosen. We choose r_e and reach $C_2 = [\,[e\,r_{ab}\,r_{be}]_u\,]_s$. Now, only two rules are applicable, $r_f \equiv [e\,\neg p_f \rightarrow f\,r_{ef}]_u \in R_2$ and $r_{\underline{e}} \equiv [e\,r_{be} \rightarrow b\,p_e]_u \in R_3$. Since R_2 has priority, r_f is applied and the configuration $C_3 \equiv [\,[f\,r_{ab}\,r_{be}\,r_{ef}]_u\,]_s$ is reached. From C_3, the unique applicable rule is $r_{\underline{f}} \equiv [f\,r_{ef} \rightarrow e\,p_f]_u \in R_3$ and $C_4 \equiv [\,[e\,r_{ab}\,r_{be}\,p_f]_u\,]_s$. Notice than the application of $r_{\underline{f}}$ is an implementation of backtracking. In the configuration C_4, the current state is e and the state f is forbidden. From C_4, only $r_{\underline{e}} \equiv [e\,r_{be} \rightarrow b\,p_e]_u \in R_3$ is applicable. The application of this rule is a new step of backtracking and it leads us to the configuration $C_5 \equiv [\,[b\,r_{ab}\,p_e\,p_f]_u\,]_s$. From C_5, two rules are applicable, $r_d \equiv [b\,\neg p_d \rightarrow d\,r_{bd}]_u \in R_2$ and $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_e]_u \in R_3$. Notice that the rule $r_e \equiv [b\,\neg p_e \rightarrow e\,r_{be}]_u \in R_2$ is not applicable due to the occurrence of the inhibitor p_e in the membrane u. Since R_2 has priority over R_3, the rule r_d is applied and the configuration $C_6 \equiv [\,[d\,r_{ab}\,r_{bd}\,p_e\,p_f]_u\,]_s$ is reached. From C_6 only backtracking can be done by applying the rule $r_{\underline{d}} \equiv [d\,r_{bd} \rightarrow b\,p_d]_u \in R_3$ and reach $C_7 \equiv [\,[b\,r_{ab}\,p_d\,p_e\,p_f]_u\,]_s$. By applying now $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_b]_u \in R_3$ the configuration $C_8 \equiv [\,[a\,p_b\,p_d\,p_e\,p_f]_u\,]_s$ is obtained. From C_8 we only can apply $r_c \equiv [a\,\neg p_c \rightarrow c\,r_{ac}]_u \in R_2$ and reach $C_9 \equiv [\,[c\,r_{ac}\,p_b\,p_d\,p_e\,p_f]_u\,]_s$. From C_9 two rules are applicable, $r_g \equiv [c\,\neg p_g \rightarrow g\,r_{cg}]_u \in R_2$ and $r_{\underline{c}} \equiv [c\,r_{ac} \rightarrow a\,p_c]_u \in R_3$. Due to the priority of R_2 over R_3, r_g is applied and the configuration $C_{10} \equiv [\,[g\,r_{ac}\,r_{cg}\,p_b\,p_d\,p_e\,p_f]_u\,]_s$ is obtained. Finally, the applicable rules are $r_F \equiv [g]_u \rightarrow \lambda \in R_1$ and $r_{\underline{g}} \equiv [g\,r_{cg} \rightarrow c\,p_g]_u \in R_3$. Since R_1 has priority over R_3, the rule r_F is applied and the configuration $C_{11} \equiv [r_{ac}\,r_{cg}\,p_b\,p_d\,p_e\,p_f]_s$. No more rules are applicable and C_{11} is a halting configuration. The objects r_{ac} and r_{cg} determine a path from the initial state to the final one. Notice that the chosen rules in the non-deterministic points are crucial. From C_0 the configuration $C_3^* \equiv [r_{ac}\,r_{cg}]_s$ is reachable in three steps by applying sequentially the rules $r_c \equiv [a\,\neg p_c \rightarrow c\,r_{ac}]_u \in R_2$, $r_g \equiv [c\,\neg p_g \rightarrow g\,r_{cg}]_u \in R_2$ and $r_F \equiv [g]_u \rightarrow \lambda \in R_1$.

5 A New Solution for the N-Queens Problem

The first step for designing a new solution for the N-queens problem is to determine the space of states. We have chosen an *incremental formulation* (see [8]), which starts from the empty state and each action adds a queen to the state. This formulation reduces drastically the space of states, since a new queen added to the description of a state can be placed only in a non forbidden square. In this way, states are arrangements of k queens ($0 \leq k \leq N$), one per column in the leftmost k columns and transitions are pairs (x,y) where the state

y is the state x with a new queen is added in the leftmost empty column. Such new queen is not attacked by any other one already present on the board.

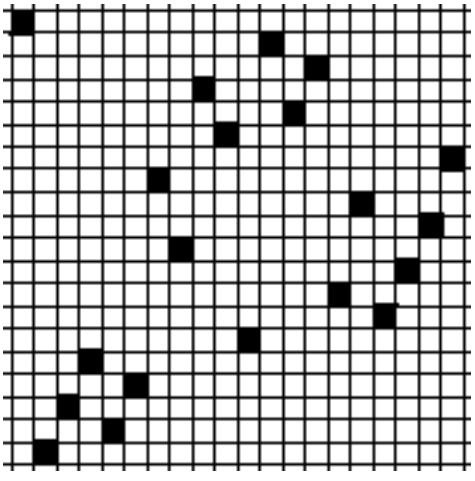
The basic idea of the P system design is to encode the position of a queen as a set of four objects x_i, y_j, u_{i-j} and v_{i+j}, where x_i represents a column and y_j represents a row ($1 \leq i, j \leq N$). The objects u_{i-j} and v_{i+j} represent the ascendant and the descendant diagonals respectively and their subindices are determined by the corresponding column and row i and j. Placing a queen on the chessboard means to choose a square, i.e., a set $\{x_i, y_j, u_{i-j}, v_{i+j}\}$ among the eligible objects and delete them from the corresponding membrane. The choice is recorded. If the final state is reached then the process finishes; otherwise we do backtracking and choose another eligible set.

We present a family of P systems which solves the decision problem associated to the N-queens problem (a P system for each value of N) slightly different from the general one presented in Section 4. We add a new set of rules R^* for removing useless objects. For each positive integer greater than 2, we consider the P system $\Pi = (\Gamma, H, \mu, w_u, w_s, R_1, R^*, R_2, R_3, R_1 > R^* > R_2 > R_3)$ where

- The alphabet $\Gamma = \{x_i, y_j, u_{i-j}, v_{i+j}, p_{i,j} : i, j \in \{1, \ldots, N\}\} \cup \{x_{N+1}\}$
- The set of labels $H = \{u, s\}$
- The initial multisets $w_u = \{x_1, y_1, \ldots, y_N, u_{1-N}, \ldots, u_{N-1}, v_2, \ldots, v_{2N}\}$ and $w_s = \emptyset$.
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- Four sets of rules R_1, R^*, R_2 and R_3
  - $R_1 = \{[x_{N+1}]_u \rightarrow \lambda : x \in F\}$. In this design, when the object k_N is reached, the membrane u is dissolved and the computation ends.
  - $R^* = \{[p_{i,j} x_{i-1} \rightarrow x_{i-1}]_u : i \in \{2, \ldots, N\}, j \in \{1, \ldots, N\}\}$ Just cleaning rules.
  - $R_2 = \{[x_i\, y_j\, u_{i-j}\, v_{i+j}\, \neg p_{i,j} \rightarrow x_{i+1}\, r_{i,j}\,]_u : i, j \in \{1, \ldots, N\}\}$ These rules put a new queen on the chessboard by choosing an eligible position.
  - $R_3 = \{[r_{i,j}\, x_{i+1} \rightarrow x_i\, y_j\, u_{i-j}\, v_{i+j}\, p_{i,j}]_u : i, j \in \{1, \ldots, N\}\}$. These rules remove one queen from the chessboard and implement the backtracking.
- Finally, the order $R_1 > R^* > R_2 > R_3$ among the sets of rules is settled.

5.1 A Brief Overview of the Computation

From the objects $\{x_1, \ldots, x_N\}$, only x_1 occurs in the initial configuration. This means that the column 1 is already chosen. In order to take the row, one of the N rules $[k_0\, x_1\, y_j\, u_{1-j}\, v_{1+j}\, \neg p_{1,j,0} \rightarrow x_2\, r_{1,j,1}\, k_2]_u$ where $j \in \{1, \ldots, N\}$ is chosen. The choice of this rule determines the square (x_1, y_j) where the first queen is placed. The application of the rule removes the objects corresponding to the column, row ascendant and descendant diagonal lines $x_1\, y_j\, u_{1-j}\, v_{1+j}$ in the chessboard. The associated column, row and diagonals to these objects are not eligible and the new queen will be placed in a *safe* square, in the sense that no other queen in the board threatens this position (i.e., there are no other queens in the same row, nor in the same column, nor in both diagonals). The application of the rule produces the object x_2. Next, a rule from the set



1-20 2-1 3-3 4-5 5-2 6-4 7-13 8-10 9-17 10- 15
11-6 12-19 13-16 14-18 15-8 16-12 17-7 18-9 19-11 20-14

Fig. 1. A solution for the 20-queens problem

$[k_1\, x_2\, y_j\, u_{2-j}\, v_{2+j}\, \neg p_{2,j,1} \rightarrow x_3\, r_{2,j,2}\, k_3]_u$ is chosen. If the successive choices are right, then the object k_N is reached and the membrane u dissolved. The objects $r_{i,j,r}$ in the membrane s from the halting configuration give us a solution to the problem. If no rules from the set R_2 can be applied, then we apply one rule from R_3. As shown in the general case, such rules implement backtracking and produce objects $p_{i,j,r}$ which act as inhibitors. Before applying rules from R_2 or R_3, the P system tries to apply rules from R_1, which means the halt of the computation, or from R^*, which clean useless inhibitor objects.

5.2 Examples

An *ad hoc* CLIPS program (available from the authors) has been written based on this design of solution for the N-queens problem based on Membrane Computing techniques. Some experiments have been performed on a system with an Intel Pentium Dual CPU E2200 at 2,20 GHz, 3GB of RAM and using CLIPS V6.241 under the operating system Windows Vista. Finding one solution took 0,062 seconds for a 4×4 board and 15,944 seconds for a 20×20 board. Figure 1 shows a solution for the 20-queens problem found by this computer program.

6 Conclusions and Future Work

The purpose of this paper is twofold. On the one hand, to stress the inviability of solutions based on brute force algorithms for intractable problems, even in case of future implementations. On the other hand, to open a door in Membrane Computing to Artificial Intelligence techniques, which are broadly studied and which can enrich the methodology of the design of P system solutions.

This first approach can be improved in many senses. As pointed out in Section 4, the aim of this paper is not minimalist and probably, searching algorithms can be implemented into P systems by using simpler P system models. The second improvement is associated to the nature of P systems. The design of P systems which compute searching is too close to the classical sequential algorithm. In fact, although the presented P system family uses non-determinism in the choice of the rules, it does not explore the intrinsic parallelism of P systems. The next step in this way is to design algorithms which use a limited form of parallelism where several rules can be applied simultaneously, but controlling the exponential explosion of brute force algorithms. The current parallel computing architectures (see, e.g., [4]) can be a clue for these new generations of *membrane algorithms*.

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

References

1. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 187–203. Springer, Heidelberg (2009)
2. Gutiérrez-Naranjo, M.A., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Solving the n-queens puzzle with P systems. In: Gutiérrez-Escudero, R., et al. (eds.) Seventh Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, Spain, vol. I, pp. 199–210 (2009)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. International Journal of Computer Mathematics 83(7), 593–611 (2006)
4. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Cecilia, J.M., Guerrero, G.D., García, J.M.: Simulation of recognizer P systems by using manycore gpus. In: Martínez-del-Amor, M.A., et al. (eds.) Seventh Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, Spain, vol. II, pp. 45–58 (2009)
5. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford Univ. Press, Oxford (2010)
6. Pérez-Jiménez, M.J.: A computational complexity theory in membrane computing. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 125–148. Springer, Heidelberg (2010)
7. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–285 (2003)
8. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2002)
9. Wikipedia, http://en.wikipedia.org/wiki/orders_of_magnitude_mass

Towards Modelling of Reactive, Goal-Oriented and Hybrid Intelligent Agents Using P Systems

Petros Kefalas and Ioanna Stamatopoulou

Department of Computer Science, CITY College, Thessaloniki, Greece,
International Faculty of the University of Sheffield
{kefalas,istamatopoulou}@city.academic.gr

Abstract. Intelligent agents are classified into various types depending on whether they just react to the stimuli they perceive (reactive) or they develop plans to solve their own goals (proactive or goal-oriented). In practice, agents are a mixture of two layers since they perform reactive or proactive tasks depending on what is the most appropriate at a given time (hybrid agents). Bearing in mind the dynamic organisation of a multi-agent system consisting of any of the above types, it is only natural to consider Population P Systems as a suitable candidate for modelling. In this paper, we describe preliminary work done towards modelling of MAS which include all types of agents. An initial attempt is made to tackle certain issues that have to do with the objects and rules that define each agent operation. Alongside the alternative solutions, we present a concrete example to demonstrate our findings and raise discussions.

1 Introduction

Intelligent agents are robotic or software entities which can exhibit autonomous, reactive, proactive and social behaviour [14]. Agents perceive their environment, react immediately if it is necessary, update their beliefs, revise their strategies, prioritise their goals and develop plans to achieve them. Multi-agent systems (MAS) are built upon the social behaviour of individual agents that can communicate, collaborate and negotiate in order to achieve their goals. Naturally, MAS are highly interactive, highly parallel and highly dynamic (change of organisation, change of roles, change in configuration etc). These dynamics make MAS specification, modelling and implementation a challenging activity. In our area of interest, formal modelling is particularly attractive as it raises many issues that cannot be tackled in a straightforward manner and leave many open challenges.

Intelligent agent architectures can be broadly categorised into reactive, goal-oriented and hybrid. In reactive agents, intelligent behaviour can be achieved without explicit symbolic representations or explicit abstract reasoning but it is an emergent property (e.g. ant colonies). The agents operation is based around a hierarchy of behaviours which resemble *if situation then action* rules.

On the other hand, goal-oriented agents and their most known representative Belief-Desire-Intention (BDI) agents [8], are based on: (a) Beliefs, i.e. the information an agent has about the environment, which may be false; (b) Desires,

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 265–272, 2010.

i.e. the things that the agent would like to see achieved; and (c) Intentions, i.e. the goals that the agent is committed to. In principle, a BDI agent perceives its environment and updates its beliefs. Based on the current state of affairs, it may revise its options and prioritise its goals. Having picked up a current goal, generates a sequence of actions that achieve the goal and executes this plan. Finally, in most cases it is necessary for the agents to exhibit both reactive and proactive behaviour, hence the hybrid model.

Practically, BDI agents are not as complicated as the underlying theory dictates [3]. Desires play a strategic role to problem solving and in highly specialised agents desires are shrink down to one, the general raison d'etre of the agent. Plans are not generated but are ready made, residing in a library of plans that are brought into the play according the current goal. The current goal is the intention that is picked up for deliberation; if it is directly executable the agent performs an action; if not, the current intention is replaced by a more analytical list of new intentions (the plan).

There have been several attempts to formally model individual intelligent agents as well as MAS structure and change. Most of them were based on state machines and their variations, but were primarily concerned with simple reactive agents [4,2]. Though such methods are adequate for the representation of the internal state of an agent, problems arose when having to deal with the dynamics of the structure of a system consisting of multiple agents. As a result other attempts used new computing paradigms, such as membrane computing. Such methods can efficiently address the aforementioned limitation of state-based methods (Population P Systems for example are very flexible in representing the dynamics of a population's structure), but were primarily concerned with biologically inspired or biological agents exhibiting emergent behaviour [10]. Finally, previous work has demonstrated that we can combine the above in order to take advantage of the complementary characteristics of the aforementioned formal methods [12]. For a complete review of this work, in terms of rationale and results, the interested reader is referred to [5].

With this paper, we initialise an effort towards modelling of goal-oriented agents using a variation of Population P Systems [1]. The following sections describe the modelling toolkit that should be available in order to formally model MAS that consists of reactive, goal-oriented or hybrid agents. We discuss the proposal along side with a MAS case in order to clarify our claims. Finally, we reach an initial definition of a Population P System suitable for modelling any type of MAS.

2 A MAS Scenario Including Goal-Oriented Agents

Assume a disaster area with civilians injured who are incapable of helping themselves in between obstacles and ruins [9]. A number of agents (rescue units or RU) are equipped with the necessary first aid kit and could provide help to injured civilians, thus temporarily rescuing victims from immediate danger. They can then broadcast the exact coordinates to the agents in their neighbourhood

and continue their rescue mission. Another set of agents (ambulance vehicles or AV) are capable of approaching the temporarily rescued civilians and carry them to a more secure establishment (e.g. emergency room or ER). Of course various parameters play an important role in this rescue scenario, such as number of agents, the amount of supplies, the capacity of the ambulances, etc. Also, one should take into account possible failures of agents as well as non-trivial interaction and mode of communication.

The development of such MAS would normally involve two kind of agents. RU would be reactive agents which would function under certain rules obeying a strict hierarchy such as for example:

if there is an obstacle then avoid obstacle $\succ$
if injured civilian is detected then
provide first aid to victim and inform nearby agents about location $\succ$
if empty space then move randomly

Injured civilians could also be modelled as reactive agents. On the other hand, AV would be goal-oriented agents which need to form plans to satisfy their goals, i.e. having updated their beliefs on where the victims are located based on incoming information and develop a sequence of actions to pick up their victims. In reality, AV agents should also have a reactive layer on top, which will respond to immediate threats, such as:

if there is an obstacle then avoid obstacle $\succ$
if at ER then upload the injured civilians $\succ$
if load reached the maximum capacity then move towards the ER $\succ$
if injured civilian is detected and not at ER then pick up victim

The above reactive layer deals with the simple behaviours, apart from *moving towards the victim* behaviour which requires planning. This is the main difference from agent RU that searches the space randomly for locating injured civilian.

Therefore the requirements for modelling the above is summarised in the following:

- modelling of individual separate agents of various types is necessary;
- the agent models should be developed with non-trivial data structures and their accompanying operations;
- there must be a way to code the rules for behaviours within an agent, including the communication behaviour;
- it is essential to set up priorities on these behaviours for the agent to perform the desired overall task;
- describing the change in communication links is desirable according to some "neighbouring" criteria;
- modelling of agents roles, generation and destruction must be possible in order to model the dynamic configuration of the system;
- agents in a MAS could operate in parallel exhibiting an asynchronous behaviour;

Most of the above naturally lead to considering elements of Tissue P Systems [7], Populations P Systems [1] and P colonies [6], equipped with some new features that could make them more focused to the modelling of goal-oriented and hybrid agents.

3 Formal Modelling of MAS

3.1 Agents as Cells

Each agent can be directly mapped to a cell. Cells are arranged in a graph (population) rather than a hierarchical tree structure. Cells must have types each corresponding to a different role of each agent in the MAS. For instance, in the rescue scenario there are four types of cells, namely a rescue unit (RU), an ambulance vehicle (AV), the civilian victim (CV) and the emergency room (ER). The latter could be an agent if it has certain characteristics (e.g. it is a mobile emergency unit, it can communicate with other agents etc.) or can be modelled as a simple entity otherwise. Instances of all these cell types make a MAS configuration. The graph denotes the communication between cells, e.g. neighbouring RUs and AVs have a direct communication, as well as a CV with a RU or AV on the same spot, the AV when it reaches the ER, the ERs between them etc.

3.2 Data Structures and Objects

Objects within cells should be more than multisets of symbols. Practically, more sets and mathematical structures are needed, such as naturals, reals, atoms, n-tuples, sequences/lists, sets, etc. as well as all operations applied on these. In addition, objects should be partitioned in various subsets, in a kind of annotated values of attributes. The absolutely necessary subsets of objects required for a goal-oriented agent are: (a) a set of Beliefs, (b) a set of Goals and (d) a set of internal agent States, and (e) a set of incoming Messages. Thus, for instance:

$B = \{(victim\_at\ X\ Y), (er\_at\ X\ Y), \ldots\}$ where $X,\ Y \in N$,
$G = \{(pickup\_victim\ X\ Y),\ (move\_towards\ X\ Y),\ (leave\_victim\_at\_er)\ldots\}$,
$States = \{doing\_nothing,\ rescuing,\ moving\_to\_er, \ldots\}$,
$IncomingMessage = \{(found\_victim\_at\ X\ Y), \ldots\}$ etc.

These in turn would be used in the list of goals, queue of incoming messages etc. Depending on the problem, some more sets might be necessary, such as the current position and direction is space, the capacity of an AV, the current load, the current supplies, fuel, etc. In practical modelling, we would need some sort of notation that differentiates objects according to the set they belong, for example:

$B : (victim\_at\ 3\ 8),\ State : rescuing,\ Pos : (4\ 5)$,
$ListOfGoals :\ \langle(move\_towards\ 6\ 7), (leave\_victim\_at\_er), \ldots\rangle$,
$IncomingQueue : \langle(found\_victim\_at\ 6\ 7),\ (found\_victim\_at\ 9\ 1), \ldots\rangle$, etc.

3.3 Behaviours and Rewrite/Communication Rules

Reactive behaviours can be modelled as a set of transformation rules for a specific agent type. For example, the rule:

$avoid\_obtacle : (State : moving\_to\_er\ Obstacle : (X\ Y)\ Pos : (X_1\ Y_1)\ Direction : D$ if $(next\_to\ X\ Y\ X_1\ Y_1) \rightarrow$
$State : moving\_to\_er\ Obstacle : (X\ Y)\ Pos : (X_1\ Y_1)\ Direction : D'$ where $(random\ D'))_{AV}$

The objects on the left hand side are consumed and replaced by the objects of the right hand side within a cell of type AV. Checks and new values are performed and produced through the guards following the *if* delimiter and operations following the *where* delimiter. All rules could be identified by a unique identifier at the far left, in this case *avoid_obtacle*.

Proactive behaviours, such as updating beliefs, adding goals to the list or executing primitive goals (actions) can be modelled in a similar way.

Similarly, there exist communication rules that are used to pass messages to cells that are linked through the graph structure. For example, the rule:

$send\_victim\_position :$
$(B : (victim\_at\ 4\ 2);\ incoming\_message : (found\_victim\_at\ 4\ 2),\ broadcast)_{RU}$

means that in the presence of an object $B : (victim\_at\ 4\ 2)$ inside a cell of type RU an object $incoming\_message : (found\_victim\_at\ 4\ 2)$ can be obtained by all neighbouring cells, that is, those connected through links. The receiving agent can put the incoming message in the queue of incoming messages through a simple transformation rule. Guards and new values may also be required in the form of the rules.

Communication rules could also be used for perceiving the environment. For instance, the rule:

$perceive\_obstacle :$
$Pos : (X\ Y);\ Obstacle : (X_1\ Y_1)$ if $(within\_range\ X\ Y\ X_1\ Y_1),\ perceive)_{AV}$

is the same as the above with the exception that object $Obstacle : (X_1\ Y_1)$ is obtained by the environment. The other way round, i.e. an object is can be expelled out to the environment through an *output* rule. The performatives *broadcast*, *perceive* and *output* are equivalent to the commonly used in membrane computing *in*, *enter* and *exit*.

3.4 Priorities of Behaviours

In order to achieve the correct overall agent behaviour, individual behaviours including communication should be ordered. A top level ordering should determine which type of task behaviour should be tried first and whether communication (either *broadcast* or *perceive* or *output*) should precede or follow task behaviours. For example, a possible ordering might be:

broadbast rules $\preceq$ *perceive* rules $\prec$ *reactive* rules $\prec$
proactive rules $\preceq$ *output* rules

which implies that incoming message and perceptions should fire first followed by reactive rules followed by proactive rules and sending out messages.

At a lower level ordering between rules within the same type must exist. For instance, a possible ordering for the reactive behaviour should be:

$$avoid\_obstcle \prec upload\_victims \prec move\_towards\_er \prec pick\_up\_victim$$

It should be noted at this point that imposing such priorities in types of rules, or rules themselves, in combination to the use of lists for objects may have a restrictive effect on the maximal parallelism of the P system, which however for MAS modelling purposes is acceptable.

3.5 Communication Links and Bond Making

The graph connecting the cells is by no means fixed. Depending on the problem, the notion of "neighbourhood" between cells can be defined. This will allow cells to communicate directly through the existing links. Establishment of links is governed by a bond making rule that states the preconditions which must be true. For example, a bond making rule between an AV and a RU can be:

$$connect\_neighboring\_agents : (AV,\ Pos : (X_{av}\ Y_{av});\ Pos : (X_{ru}\ Y_{ru}), RU)$$
$$\text{if } (neighbours\ X_{av}\ Y_{av}\ X_{ru}\ Y_{ru})$$

meaning that in the presence of neighbouring $Pos : (X_{av}\ Y_{av})$ and $Pos : (X_{av}\ Y_{av})$ inside two cells of type AV and RU respectively, a bond is created between the two cells.

3.6 Dynamic Structure and Cell Differentiation/Division/Death

It is often the case that new agents should appear into the system, perhaps some change role and eventually some will disappear. Such situations can be handled effectively by cell division, cell differentiation and cell death rules. For instance, a RU which runs out of fuel is removed from the system:

$$out\_of\_order : (fuel : 0)_{RU} \rightarrow \dagger$$

4 Main Proposal

Bearing in mind the above, we propose that MAS consisting of any type of agents can be modelled using a variation of Population P Systems with Active Membranes, which can be defined as the construct:
${}^{\alpha}\mathcal{P} = (V, \Phi, T, \gamma, \alpha, w_E, A_1, A_2, \ldots, A_n, R_b, R_s, O_b, O_r, O_p)$ where:

- V is a finite set of structures and symbols called objects and $V = Beliefs \cup Goals \cup States \cup Messages \ldots$;
- Φ is a set of default and user-defined operations on items of V;
- T is a finite alphabet of symbols, which define different types of agents;
- $\gamma = (\{1, 2, \ldots n\}, G)$, with $G \subseteq \{\{i, j\} \mid 1 \leq i \neq j \leq n\}$, is a finite undirected graph;
- α is a finite set of bond-making rules;

- $w_E \in V^*$ is a finite multi-set of objects initially assigned to the environment;
- $A_i = (w_i, t_i)$, for each $1 \leq i \leq n$, with $w_i \in V^*$ a finite multi-set of objects, and $t_i \in T$ the type of agent/cell i;
- R_b is a finite set of behavioural rules and $R_b = BroadCast \cup Perceive \cup Output \cup Reactive \cup Proactive$.
- R_s is a finite set of structural rules and $R_s = Differentiation \cup Division \cup Death$.
- O_b is a partial order over behaviours R_b
- O_r is a partial order over the set of *Reactive* behaviour rules
- O_p is a partial order over the set of *Proactive* behaviour rules

The form of the rules is not formally defined here but can be fairly directly implied by the examples stated above.

5 Conclusions and Open Issues

We have made an initial attempt to define a new variation of Population P Systems with Active Membranes, namely ${}^{\alpha}\mathcal{P}$, which is suitable for modelling multi-agent systems including all types of agents, such as reactive, goal-oriented and hybrid. This is the main difference from previous work, in which we only dealt with simple biological reactive agents. We demonstrated the need through an example of a disaster scenario in which agents are trying to rescue civilians injured. A more concrete and precise definition, including the theoretical BDI model as well as the detailed computation steps are in our immediate intentions. Before that, however, we need to identify the other practical issues raised by such modelling. The main question is whether the constructs of the ${}^{\alpha}\mathcal{P}$ are adequate to map a MAS (including BDI agents) or there is a need to extend it with new ones. This will lead us to the design and implementation of a tool that animates the models, along the lines of previous work done both textually [11] and visually [13].

References

1. Bernardini, F., Gheorghe, M.: Population P Systems. Journal of Universal Computer Science 10(5), 509–539 (2004)
2. Coakley, S.: Formal Software Architecture for Agent-Based Modelling in Biology. PhD thesis, Dept. of Comp. Science, Univ. of Sheffield, UK (2007)
3. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: Proc. of the 6th Conference on Artificial Intelligence, pp. 677–682 (1987)
4. Kefalas, P., Holcombe, M., Eleftherakis, G., Gheorghe, M.: A formal method for the development of agent-based systems. In: Plekhanova, V. (ed.) Intelligent Agent Software Engineering, pp. 68–98. Idea Publishing Group Co., USA (2003)
5. Kefalas, P., Stamatopoulou, I.: Modelling of multi-agent systems: Experiences with membrane computing and future challenges. In: Applications of Membrane computing, Concurrency and Agent-based modelling in POPulation biology (AMCA-POP), Satellite event of the 11th Conference on Membrane Computing (to appear, 2010)

6. Kelemen, J., Kelemenova, A., Paun, G.: Preview of P colonies: A biochemically inspired computing model. In: Pollack, J.B., Bedau, M., Husbands, P., Ikegami, T., Watson, R.A. (eds.) Proceedings of the 9th Intern. Conference on the Simulation and Synthesis of Living Systems (Alife IX), pp. 82–86. MIT Press, Cambridge (2004)
7. Martin-Vide, C., Păun, G., Pazos, J., Rodriguez-Paton, A.: Tissue P systems. Theoretical Computer Science 296, 295–326 (2003)
8. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: Allen, J., Fikes, R., Sandewall, E. (eds.) Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR 1991), pp. 473–484. Morgan Kaufmann, San Francisco (1991)
9. Sakellariou, I., Kefalas, P., Stamatopoulou, I.: Enhancing NetLogo to simulate BDI communicating agents. In: Darzentas, J., Vouros, G.A., Vosinakis, S., Arnellos, A. (eds.) SETN 2008. LNCS (LNAI), vol. 5138, pp. 263–275. Springer, Heidelberg (2008)
10. Stamatopoulou, I., Gheorghe, M., Kefalas, P.: Modelling dynamic configuration of biology-inspired multi-agent systems with Communicating X-machines and Population P Systems. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 389–401. Springer, Heidelberg (2005a)
11. Stamatopoulou, I., Kefalas, P., Eleftherakis, G., Gheorghe, M.: A modelling language and tool for Population P Systems. In: PCI 2005 (2005b)
12. Stamatopoulou, I., Sakellariou, I., Kefalas, P., Eleftherakis, G.: OPERAS for social insects: Formal modelling and prototype simulation. Special Issue of Romanian Journal of Information Science and Technology (ROMJIST) on Natural Computing — from biology to computer science and back to applications 11(3), 267–280 (2008)
13. Wilensky, U.: Netlogo Center for Connected Learning and Computer-based Modelling. Northwestern University, Evanston, IL (1999), http://ccl.northwestern.edu/netlogo
14. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10(2), 115–152 (1995)

Goldbeter's Mitotic Oscillator Entirely Modeled by MP Systems

Vincenzo Manca and Luca Marchetti

University of Verona, Department of Computer Science
Strada Le Grazie 15, 37134 Verona, Italy
vincenzo.manca@univr.it, luca.marchetti@univr.it

Abstract. MP systems are a class of P systems introduced for modeling metabolic processes. Here we apply an algorithm, we call Log-Gain Stoichiometric Stepwise Regression (LGSS), to Golbeter's oscillator. In general, LGSS derives MP models from the time series of observed dynamics. In the case of Golbeter's oscillator, we found that by considering different values of the resolution time τ, different analytical forms of regulation maps were appropriate. By means of a suitable MATLAB implementation of LGSS, we automatically generated 700 MP models (τ varying from 10^{-3} min to $700 \cdot 10^{-3}$ min with increments of 10^{-3} min). Many of these models exhibit a good approximation, and have second degree polynomials as regulation maps. These results provide an experimental evidence of LGSS adequacy.

1 Introduction

Any living organism has to maintain processes which: i) introduce matter of some kinds from the external environment, ii) transform internal matter by changing the molecule distribution of a number of biochemical species (substances, metabolites), and iii) expel matter that is not useful or dangerous to the organism. The molecule distribution identifies the metabolic state of the system in question, and can be represented as a multiset over a set of molecular species.

An important problem of systems biology is the mathematical definition of a dynamical system that explains the observed dynamics of a phenomenon under investigation, by taking into account what is already known about the phenomenon. When this is possible, then we can hope that a greater knowledge of the phenomenon is gained.

An important line of research of biological modeling is aimed at defining new classes of discrete models avoiding some limitations of classical continuous models based on ordinary differential equations (ODE). In fact, very often, the evaluation of the kinetic reaction rates in differential models is problematic because it may require measurements hardly accessible in living organisms. Moreover, these measurements dramatically alter the context of the investigated processes. In contrast to ODEs, Metabolic P systems (MP systems) [11,9,8,10], based on Păun's P systems [14], were introduced for modeling *metabolic systems* by means of suitable multiset rewriting grammars.

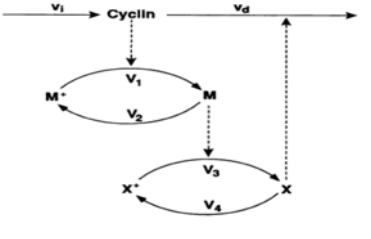
M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 273–284, 2010.

In MP systems no single instantaneous kinetic is addressed, but rather the variation of the whole system under investigation is considered, at discrete time points, separated by a specified macroscopic interval τ. The dynamics is given along a sequence of steps and, at each step, it is governed by partitioning the matter among reactions which transform it. The log-gain theory of MP systems [8] is aimed at reconstructing the *flux regulation maps* associated to the metabolic transformations. Metabolic P systems proved to be promising in many contexts and their applicability was tested in many situations where differential models are prohibitive due to the unavailability or the unreliability of the kinetic rates [10,12].

Here we apply an algorithm, we call Log-Gain Stoichiometric Stepwise Regression (LGSS), to Golbeter's oscillator given in Table 1 [3,4,5]. In this manner, we generate automatically 700 models of this oscillator, which, for the most part, provide the same order of approximation of Golbeter's model. Moreover, by considering the phenomenon at different time grains, we obtain different models and in many cases the analytical form of these models is simpler than Golbeter's model.

The fundamental mechanism of mitotic oscillations concerns the periodic change in the activation state of a protein produced by the *cdc2* gene in fission yeast or by homologous genes in other eukaryotes. The simplest form of this mechanism is found in early amphibian embryos (see [5] at page 24). Here (see the picture in the left part of Table 1) cyclin (C) is synthesized at a constant rate and triggers the transformation of inactive (M^+) into active (M) *cdc2* protein, which leads to the formation of a complex known as M-phase promoting factor (*MPF*). MPF triggers mitosis, but at the same time M elicits the activation of a protease from state X^+ to X. The active protease then degrades cyclin resulting in the inactivation of *cdc*2. This brings the cell back to initial conditions and a new division cycle can take place. The ODE presented in the right part of Table 1 is the differential model of dynamics described in the left part of Figure 1 at page 277, where C, M, X are the concentrations of C, M, X respectively and $1-M, 1-X$ are the concentrations of M^+, X^+ respectively (the definitions of the parameters of the ODE model of table 1 are not simple and are not relevant for our further discussion, however they can be found in [3]).

Table 1. Goldbeter's oscillator, which has a cycle of about 25 min [3]



$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_d + C} - k_d C$$
$$\frac{dM}{dt} = V_{M1} \frac{C}{K_c + C} \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M}$$
$$\frac{dX}{dt} = M V_{M3} \frac{(1-X)}{K_3 + (1-X)} - V_4 \frac{X}{K_4 + X}$$

2 MP Systems

Metabolic P systems, or P metabolic systems, represent metabolic processes in a discrete mathematical framework. The letter P of MP systems comes from the theoretical framework of P systems introduced by Gheorghe Păun [14] in the context of membrane computing. In fact MP systems are a special class of P systems introduced in 2004 [11] to express metabolism in a discrete mathematical setting.

A metabolic P system is essentially a multiset grammar where multiset transformations are regulated by functions. Namely, a multiset rule like $A + B \to C$ means that a number u of molecules of kind A and the same number u of molecules B are replaced by u molecules of type C. The value of u is the *flux* of the rule application. Assume to consider a system at some time steps $0, 1, 2, \ldots, t$, and consider a substance x that is produced by rules r_1, r_3 and is consumed by rule r_2. If $u_1[i], u_2[i], u_3[i]$ are the fluxes of the rules r_1, r_2, r_3 respectively, in the passage from step i to step $i + 1$, then the variation of substance x is given by:

$$x[i + 1] - x[i] = u_1[i] - u_2[i] + u_3[i] \tag{1}$$

In a MP system it is assumed that in any state the flux of each rule is provided by a function, called *regulator* of the reaction. Substances, reactions, and regulators (plus parameters which are variables different from substances occurring as arguments of regulators) specify a discrete dynamics at steps indexed in the set $\mathbb{N}$ of natural numbers. Moreover, a *temporal interval* τ, a conventional *mole size* ν, and substances masses are considered, which specify the time and population (discrete) granularities respectively. They are *scale factors* that do not enter directly in the definition of the dynamics of a system, but are essential for interpreting it at a specific physical level of mass and time granularity.

From a mathematical point of view, a MP system M of type (n, m, k), that is, of n substances, m reactions and k parameters (this type will be implicitly assumed), is specified as follows (see also [10]):

Definition 1 (MP system). *A MP system is a discrete dynamical system specified by a construct*

$$M = (S, R, H, \Phi, \tau, \nu, \mu)$$

where S, R are finite disjoint sets and the following conditions hold ($n, m, k \in \mathbb{N}$):

- *S is a set of n substances (the types of molecules) determining, for any metabolic state of the system, a vector X of substance quantities which varies on $\mathbb{R}^n$;*
- *R is a set of m reactions specified by m pairs $(r_1^-, r_1^+), \ldots, (r_m^-, r_m^+) \in \mathbb{N}^n \times \mathbb{N}^n$, composed by the left and right vectors of the reactions (relative to the reactants and to the products respectively). The matrix $\mathbb{A} = (r_1^\#, \ldots, r_m^\#)$ is the stoichiometric matrix associated to the reactions having as columns the stoichiometry balances of the rules;*
- *$H : \mathbb{N} \to \mathbb{R}^k$ is a function providing, at each step $i \in \mathbb{N}$, the vector $H[i]$ of parameters;*

- $\Phi = (\varphi_1, \ldots, \varphi_m)$ *is a vector of regulators (or flux regulation functions), where* $\Phi : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m$ *provides the fluxes of reactions corresponding to any global state of the system, that is, a pair in* $\mathbb{R}^n \times \mathbb{R}^k$ *constituted by the metabolic state and by the parameter vector. Given a reaction* r*, the substances and the parameters which occur as arguments of the corresponding regulator* φ_r *are called the tuners of the reaction* r*.*
- $\tau \in \mathbb{R}$ *is the time interval between two consecutive steps;*
- $\nu \in \mathbb{R}$ *is the number of molecules which gives a (conventional) mole in the model;*
- $\mu \in \mathbb{R}^n$ *is the vector of the mole masses of substances.*

Given a vector $X[0] \in \mathbb{R}^n$ *relative to an initial state of a given system, the dynamics of* M *is specified by the following vector recurrent equation, called* $EMA[i]$ *(Equational Metabolic Algorithm), where* $\times$ *is the usual matrix product, and, in dependence on the context,* $+$ *is the usual sum or the component-wise vector sum:*

$$X[i+1] = \mathbb{A} \times U[i] + X[i] \tag{2}$$

providing the state of the system $X[i+1]$*, for each step* $i \in \mathbb{N}$*, by means of the vector of fluxes* $U[i] = (u_r[i] \mid r \in R)$ *where* $u_r[i] = \varphi_r(a[i], b[i], \ldots)$ *and* $a[i], b[i], \ldots$ *are components of* $X[i]$*,* $H[i]$ *which are the tuners of reaction* r*.*

A MP system is completely described by a *MP grammar* where multiset rewriting rules (reactions) are given with the corresponding regulators (plus parameter evolution functions and scale factors for a complete specification of the system). A MP grammar can be also specified by a *MP graph* where the relationships between reactions and regulators appear in a more direct way. An example of MP graph, which represents the Golbeters oscillator [3], is given in the right part of Figure 1.

A Java software, called *MetaPlab*, was developed starting from a prototypal version. MetaPlab is downloadable from the official site of MetaPlab software[1]. This platform enables the user to design MP models by means of some useful graphical tools, to simulate their dynamics, and to automatise some procedures which can help the user to develop new models. MetaPlab is based on an extensible set of plugins, namely Java tools, for solving specific tasks relevant in the framework of MP systems. A guide for this software is available at the official site of MetaPlab software.

2.1 The Log-Gain Principle of MP Systems

The log-gain principle was introduced in MP systems theory for solving the following inverse dynamic problem [8,10]. Given a time series $(X[i], H[i]) \in \mathbb{R}^{n+k}$ (for $i = 0, 1, 2, \ldots t$) of some consecutive states and parameters of a metabolic system (at a time interval τ), is it possible to deduce a corresponding time series of vectors $U[i] \in \mathbb{R}^m$ which put in the equation (2) provide the time series of substance quantities? This is the dynamical problem of reaction flux

[1] `http://mplab.scienze.univr.it`

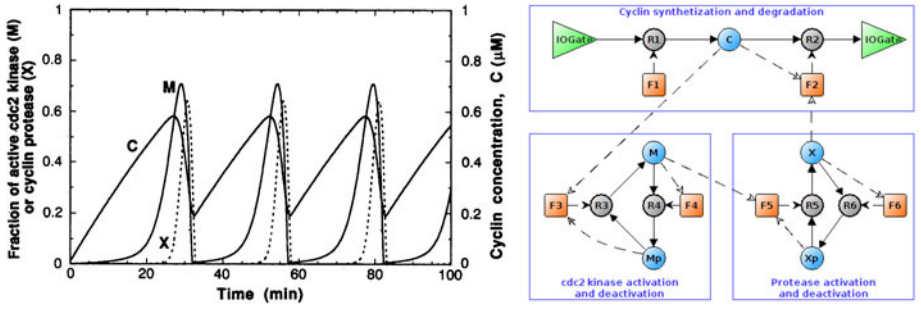


Fig. 1. On the left: a numerical solution of the set of differential equations (right part of Table 1) comprising the model introduced by A. Goldbeter (figure taken from [3]). On the right: a MP graph which represents the Golbeter's oscillator. Nodes: triangles represent matter introduction and expulsion, circles C, M, Mp, X, Xp stand for substances, circles $R1, R2, \ldots, R7$ for reactions, rounded corner rectangles for regulators, and rectangles for parameters. Edges: transformation edges go from substances to reactions (consumption) and from reactions to substances (production), regulation edges go from regulators to reactions, and influence edges go from substance or parameters (tuners) to regulators.

discovery. The deduction of time series $U[i]$ is related to the time granularity τ of the systemic logic governing the matter transformations of the observed metabolic states. When vectors $U[i]$ are known, the discovery of maps Φ which provide $U[i]$, in correspondence to the vectors $(X[i], H[i])$, is a typical problem of approximation which can be solved with standard techniques of mathematical regression.

An important remark is due in this context. The approach of flux discovery is essentially observational, macroscopic, and global, in a sense which is opposite to the perspective of differential models, which is infinitesimal, and local. In fact, we do not intend to discover the real kinetics responsible for the biochemical dynamics of each reaction, but we only try to capture the global pattern of reaction ratios of an observed dynamics. In other words, leaving unknown the *real* local internal dynamics, we decide to consider the system at an abstraction level which is sufficient to reveal the logic of the behavior we observe. This more abstract approach can be less informative, with respect to specific important details, but such a more generic information could be very useful in discriminating important aspects of the reality, and often, especially in the case of very complex systems, is the only way for grasping a kind of comprehension of the reality under investigation.

We call the system (2) ADA (Avogadro and Dalton Action), when we search to determine $U[i]$ from the knowledge of substance quantities (Avogadro refers to the integer stoichiometric coefficients, and Dalton to the summation of the effects of reactions). The log-gain principle assists us by adding new knowledge to the stoichiometric information of ADA equations. This principle derives from

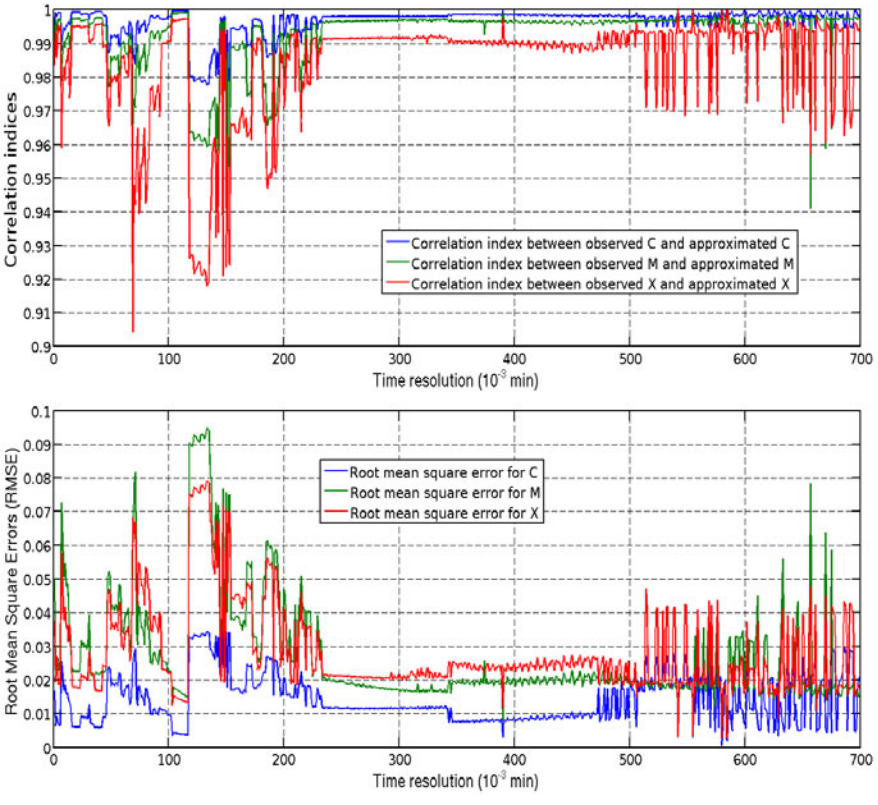


Fig. 2. Correlation indices of the three substances for each model w. r. t. the values of resolution time τ (on the top). Root mean square errors (RMSE) of the three substances for each model w. r. t. the values of resolution time τ (on the bottom).

a general biological principle called *allometry* [1], according to which, in a living organism, the global variation of its typical variables are proportional to the relative variations of the variables related to them. In differential terms the relative variation in time of a variable coincides with the variation of its logarithm, therefore we used the term "log-gain" for any law grounded on this assumption. In the specific context of our problem, we assume that *the relative variation of a reaction flux is a linear combination of the relative variations of substance quantities and parameters affecting the reaction.* We refer to the papers [8,10] for a detailed account on the log-gain theory of MP systems.

The *Log-Gain Stoichiometric Stepwise regression* algorithm (LGSS), presented and motivated in [13], combines and extends the log-gain principle with the classical method of *Stepwise Regression* [6,2], which is a statistical regression technique based on *Least Square Approximation* [7,12] and a *Fisher test F*. In fact stepwise regression tries to find the best combination of some prefixed basic functions for approximating a given time series. In LGSS we add the specific

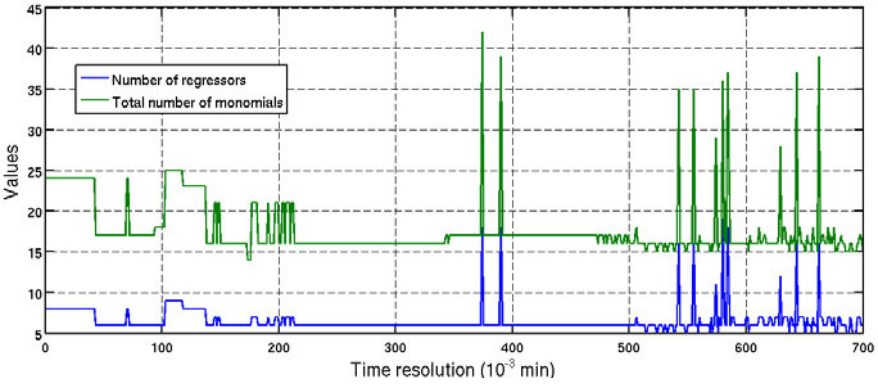


Fig. 3. Plot of the number of different regressors and of the total number of monomials for each model w. r. t. the values of resolution time τ

knowledge of the stoichiometry of the system under investigation and the requirement that the log-gain principle has to be satisfied in the best possible way. We do not give here the details of the algorithm, which was implemented by suitable MATLAB functions, but it turned definitively out that the addition of these two aspects, related to the particular nature of metabolism, provided an effective improvement of the approximation performance of stepwise regression.

3 Statistical Distribution of Mitotic MP Models

In general, LGSS derives MP models from the time series of observed dynamics. In the case of Golbeter's oscillator [3,4,5] we found that by considering different values of the resolution time τ, different analytical forms of regulation maps were appropriate. By means of a suitable MATLAB implementation of LGSS, we automatically generated 700 MP models (τ varying from 10^{-3} min to $700 \cdot 10^{-3}$ min with increments of 10^{-3} min). Figure 2 displays Pearson's correlation indices [16] and root mean square errors (RMSE) for each of these models. Each model provides an RMSE with magnitude order at most equal to 10^{-2} and permits the calculation of values which are highly correlated to the observed one.

The regulation maps calculated by the LGSS are obtained starting from a dictionary of 20 possible *regressors*, that is monomials of C, M and X with degree less than or equal to 3 (i.e. the constant, C, M, X, C^2, M^2, X^2, CM, CX, MX, C^3, M^3, X^3, C^2M, CM^2, C^2X, CX^2, M^2X, MX^2 and CMX)[2]. Figure 3 displays a diagram giving the number of regressors and the total number of monomials occurring in all the regulation maps of each MP model. We need at least 6 different regressors to get a good MP model while we need at least 18 monomials to define all the regulation maps that comprise a model.

[2] Substances M^+ and X^+ are not considered because they depend on M and X respectively.

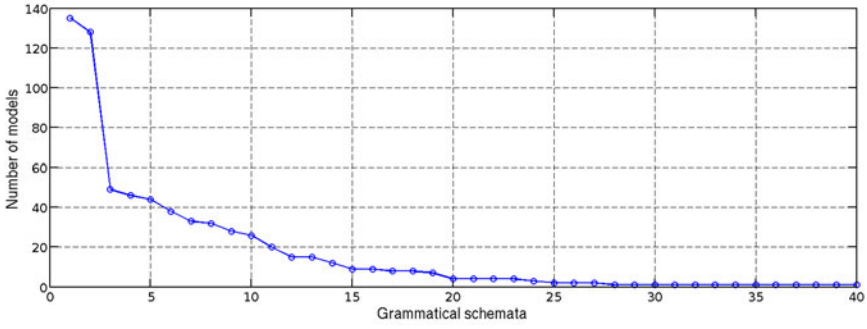


Fig. 4. Number of MP models w. r. t. grammatical schemata (please see Table 2 for details)

4 Model Classification According to Descriptional Parameters

Given a MP grammar, providing the dynamics of a MP model, we can abstract from the particular values of the constants of regressors by identifying a MP grammatical schemata defined in terms only of the analytical form of regressors constituting each regulation map. In this section we present the distribution of these grammatical schemata over the population of mitotic models. We found that all the 700 models are distributed into 40 different grammatical schemata. If we order these schemata according to the number of models where they occur we find the distribution given in Figure 4. For example the grammatical schemata at the 10th position has 26 models. In Table 2 other descriptional indices of models are given for the first 14 grammatical schemata which define 621 models from a total of 700 (89%). These indices are useful for discriminating interesting aspects of the MP grammars and they comprehend:

1. the *number of regressors*;
2. the *total number of monomials*;
3. the *temporal grain* of dynamics observation which is expressed by the values of time interval τ;
4. the *best value of* τ which is relative to the model which provides the best dynamical approximation of the mitotic phenomenon;
5. the best *RMSE* which is the average value of the RMSE relative to the substances curves corresponding to the best τ.

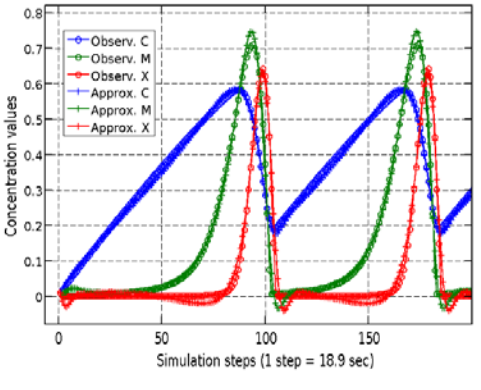
It is worthwhile to remark that the grammatical schemata occurring at the first positions (with high frequency) are also the grammatical schemata having a small number of regressors and monomials.

Table 2. Descriptional indices of models given for the first 14 grammatical schemata ordered as explained in section 4

Grammatical schemata	number of models	number of regressors	total n. of monomials	τ interval (10^{-3} *min*)	best τ (10^{-3} *min*)	best RMSE
1	135	6	16	151 – 345	315	$1.61 \cdot 10^{-2}$
2	128	6	17	343 – 477	401	$1.62 \cdot 10^{-2}$
3	49	6	17	43 – 93	43	$1.84 \cdot 10^{-2}$
4	46	6	16	138 – 232	219	$1.95 \cdot 10^{-2}$
5	44	8	24	1 – 71	40	$1.48 \cdot 10^{-2}$
6	38	6	16	525 – 699	683	$1.78 \cdot 10^{-2}$
7	33	6	16	473 – 563	556	$1.79 \cdot 10^{-2}$
8	32	5	15	514 – 694	602	$2.78 \cdot 10^{-2}$
9	28	7	16	570 – 696	671	$1.09 \cdot 10^{-2}$
10	26	6	16	493 – 684	684	$1.8 \cdot 10^{-2}$
11	20	8	23	118 – 137	137	$5.86 \cdot 10^{-2}$
12	15	9	25	103 – 117	103	$9.6 \cdot 10^{-3}$
13	15	6	17	474 – 499	474	$1.62 \cdot 10^{-2}$
14	12	7	21	191 – 212	212	$1.97 \cdot 10^{-2}$

Table 3. The best MP mitotic oscillator whose MP grammar belongs to the grammatical schemata 1 of Table 2 ($\tau = 315 \cdot 10^{-3}$ min, $RMSE \approx 1.61 \cdot 10^{-2}$). Constants and initial values: $v_i = 0.025$, $k_1 = 0.0158$, $k_2 = 0.0168923$, $k_3 = 0.0428226$, $k_4 = 0.054506$, $k_5 = 0.03327$, $k_6 = 0.0485192$, $k_7 = 0.00245843$, $k_8 = 0.540636$, $k_9 = 0.219284$, $k_{10} = 0.14129$, $k_{11} = 0.308615$, $k_{12} = 1.01307$, $k_{13} = 0.0338141$, $k_{14} = 0.468994$, $k_{15} = 0.756053$, $k_{16} = 1.15991$, $C[0] = M[0] = X[0] = 0.01$, $M^+[0] = X^+[0] = 0.99$.

$$
\begin{array}{ll}
r_1 : \emptyset \rightarrow C & \varphi_1 = v_i \\
r_2 : C \rightarrow \emptyset & \varphi_2 = k_1 + k_2\, C + k_3\, M + k_4\, X - k_5\, C^2 - k_6\, CM \\
r_3 : M^+ \rightarrow M & \varphi_3 = k_7 + k_8\, CM \\
r_4 : M \rightarrow M^+ & \varphi_4 = k_9\, M + k_{10}\, X \\
r_5 : X^+ \rightarrow X & \varphi_5 = k_{11}\, C + k_{12}\, M \\
r_6 : X \rightarrow X^+ & \varphi_6 = k_{13} + k_{14}\, X + k_{15}\, C^2 + k_{16}\, CM
\end{array}
$$



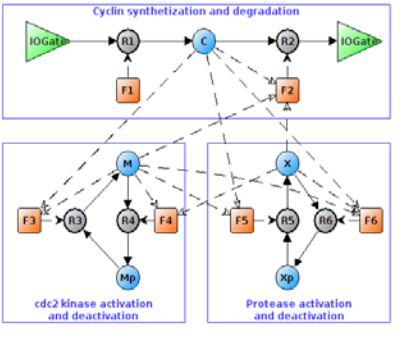


Table 4. The best MP mitotic oscillator whose MP grammar belongs to the grammatical schemata 2 of Table 2 ($\tau = 401 \cdot 10^{-3}$ min, $RMSE \approx 1.62 \cdot 10^{-2}$). Constants and initial values: $v_i = 0.025$, $k_1 = 0.0129$, $k_2 = 0.0255671$, $k_3 = 0.0666719$, $k_4 = 0.0632731$, $k_5 = 0.0522867$, $k_6 = 0.0749538$, $k_7 = 0.02125$, $k_8 = 0.836282$, $k_9 = 0.00202831$, $k_{10} = 0.385222$, $k_{11} = 0.14451$, $k_{12} = 0.392585$, $k_{13} = 1.2218$, $k_{14} = 0.0346891$, $k_{15} = 0.586917$, $k_{16} = 0.962714$, $k_{17} = 1.35871$, $C[0] = M[0] = X[0] = 0.01$, $M^+[0] = X^+[0] = 0.99$.

$$
\begin{array}{ll}
r_1 : \emptyset \to C & \varphi_1 = v_i \\
r_2 : C \to \emptyset & \varphi_2 = k_1 + k_2\, C + k_3\, M + k_4\, X - k_5\, C^2 - k_6\, CM \\
r_3 : M^+ \to M & \varphi_3 = k_7\, C + k_8\, CM \\
r_4 : M \to M^+ & \varphi_4 = k_9 + k_{10}\, M + k_{11}\, X \\
r_5 : X^+ \to X & \varphi_5 = k_{12}\, C + k_{13}\, M \\
r_6 : X \to X^+ & \varphi_6 = k_{14} + k_{15}\, X + k_{16}\, C^2 + k_{17}\, CM
\end{array}
$$

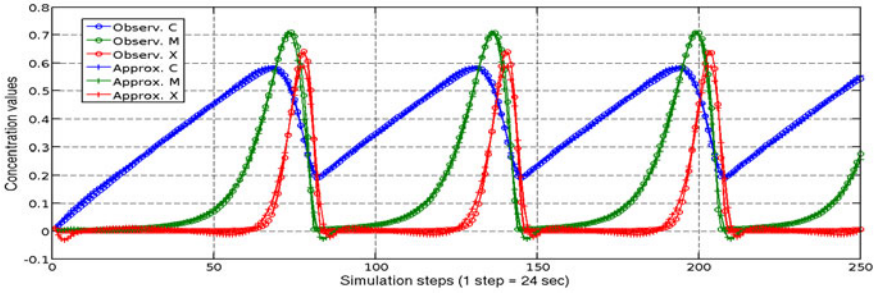


Table 5. The MP mitotic oscillator with the minimum number of different regressors ($\tau = 602 \cdot 10^{-3}$ min, $RMSE \approx 2.78 \cdot 10^{-2}$). Constants and initial values: $v_i = 0.025$, $k_1 = 0.0123$, $k_2 = 0.116301$, $k_3 = 0.0922507$, $k_4 = 0.00704311$, $k_5 = 0.148285$, $k_6 = 0.0596357$, $k_7 = 1.78159$, $k_8 = 0.0162002$, $k_9 = 0.922378$, $k_{10} = 0.119154$, $k_{11} = 0.0388314$, $k_{12} = 1.38018$, $k_{13} = 0.173718$, $k_{14} = 0.634806$, $k_{15} = 1.69501$, $C[0] = M[0] = X[0] = 0.01$, $M^+[0] = X^+[0] = 0.99$.

$$
\begin{array}{ll}
r_1 : \emptyset \to C & \varphi_1 = v_i \\
r_2 : C \to \emptyset & \varphi_2 = k_1 + k_2\, M + k_3\, X - k_4\, C - k_5\, CM \\
r_3 : M^+ \to M & \varphi_3 = k_6\, C + k_7\, CM \\
r_4 : M \to M^+ & \varphi_4 = k_8 + k_9\, M + k_{10}\, X \\
r_5 : X^+ \to X & \varphi_5 = k_{11} + k_{12}\, M \\
r_6 : X \to X^+ & \varphi_6 = k_{13}\, C + k_{14}\, X + k_{15}\, CM
\end{array}
$$

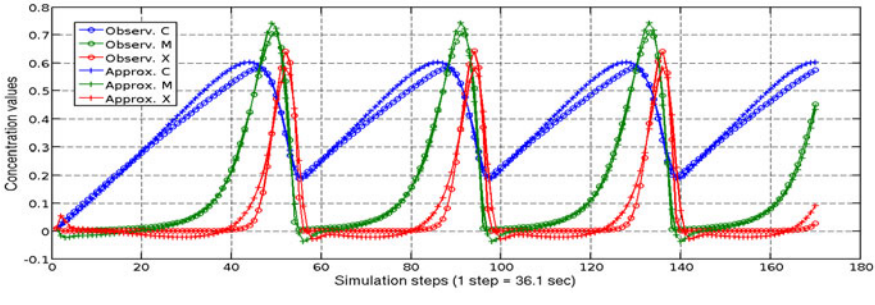
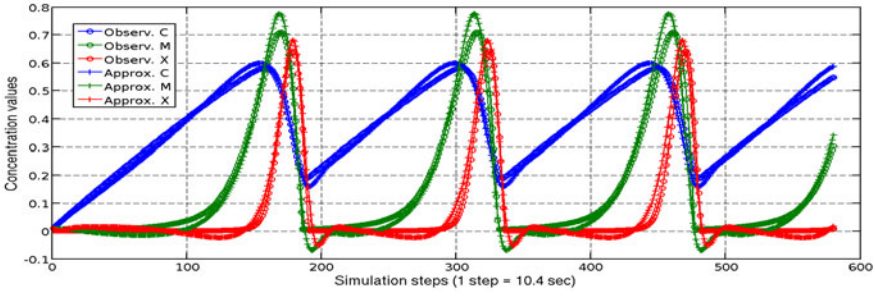


Table 6. The MP mitotic oscillator with the minimum total number of monomials ($\tau = 173 \cdot 10^{-3}$ min, $RMSE \approx 2.67 \cdot 10^{-2}$). Constants and initial values: $v_i = 0.025$, $k_1 = 0.0209$, $k_2 = 0.0149329$, $k_3 = 0.0351323$, $k_4 = 0.0200062$, $k_5 = 0.000662743$, $k_6 = 0.215816$, $k_7 = 0.0696881$, $k_8 = 0.0911799$, $k_9 = 0.166106$, $k_{10} = 0.569463$, $k_{11} = 0.00823672$, $k_{12} = 0.252676$, $k_{13} = 0.404647$, $k_{14} = 0.668527$, $C[0] = M[0] = X[0] = 0.01$, $M^+[0] = X^+[0] = 0.99$.

$$
\begin{array}{ll}
r_1 : \emptyset \to C & \varphi_1 = v_i \\
r_2 : C \to \emptyset & \varphi_2 = k_1 + k_2\ M + k_3\ X - k_4\ CM \\
r_3 : M^+ \to M & \varphi_3 = k_5 + k_6\ CM \\
r_4 : M \to M^+ & \varphi_4 = k_7\ M + k_8\ X \\
r_5 : X^+ \to X & \varphi_5 = k_9\ C + k_{10}\ M \\
r_6 : X \to X^+ & \varphi_6 = k_{11} + k_{12}\ X + k_{13}\ C^2 + k_{14}\ CM
\end{array}
$$



4.1 Analytical Forms of Mitotic MP Grammars

In this section we will present a number of MP models where regulation maps present very simple and nice forms especially in comparison with the analytical form of Goldbeter's model of Table 1. They were chosen according to some criteria of representativeness that are based on the classification analysis developed in the previous section.

Tables 3 and 4 give the best two MP mitotic oscillators whose grammars belong to the first two grammatical schemata presented in Table 2. Tables 5 and 6 give the two simpler MP grammars which define a mitotic oscillator: the first one uses the minimum number of different regressors (only 5: the constant, C, M, X and CM) while the second one uses the minimum (total) number of monomials (only 14 monomials).

5 Conclusions

In this paper, by using Golbeter's oscillator as a case study, we show that metabolic P systems yield a robust method for biological modeling. The method we used can be applied without any knowledge about reaction rate kinetics, and can provide, with respect to differential models, different and even simpler mathematical formulations. This possibility is strictly related to the chosen time scale of observed dynamics, and seems to be a promising perspective towards *multi-scale modeling*, which is a challenging aspect in systems biology.

In [13] we develop a systematic analysis and a generalization of the algorithm of Log-Gain Stoichiometric Stepwise Regression (LGSS) on which our results are based. It combines the equational formulation of MP dynamics with the log-gain principle and with the classical statistical regression technique of stepwise regression. This algorithm represents the most recent solution, in terms of MP systems, of the inverse dynamics problem, that is, of the identification of (discrete) mathematical models exhibiting an observed dynamics and satisfying all the constraints imposed by the specific knowledge about the modeled phenomenon.

References

1. von Bertalanffy, L.: General Systems Theory: Foundations, Developments, Applications. George Braziller Inc., New York (1967)
2. Draper, N., Smith, H.: Applied Regression Analysis, 2nd edn. John Wiley & Sons, New York (1981)
3. Goldbeter, A.: A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. PNAS 88(20), 9107–9111 (1991)
4. Goldbeter, A.: Biochemical Oscillations and Cellular Rhythms: The molecular bases of periodic and chaotic behaviour. Cambridge University Press, Cambridge (1996)
5. Goldbeter, A.: Computational approaches to cellular rhythms. Nature 420, 238–245 (2002)
6. Hocking, R.R.: The Analysis and Selection of Variables in Linear Regression. Biometrics, 32 (1976)
7. Luenberger, D.G.: Optimization by Vector Space Methods. John Wiley & Sons, Chichester (1969)
8. Manca, V.: Log-Gain Principles for Metabolic P Systems. In: Condon, A., et al. (eds.) Algorithmic Bioprocesses. Natural Computing Series, ch. 28, pp. 585–605. Springer, Heidelberg (2009)
9. Manca, V.: Fundamentals of Metabolic P Systems. In: [15], ch. 19.Oxford University Press, Oxford (2010)
10. Manca, V.: Metabolic P systems. Scholarpedia 5(3), 9273 (2010)
11. Manca, V., Bianco, L., Fontana, F.: Evolutions and Oscillations of P systems: Theoretical Considerations and Application to biological phenomena. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 63–84. Springer, Heidelberg (2005)
12. Manca, V., Marchetti, L.: Metabolic approximation of real periodical functions. Journal of Logic and Algebraic Programming (2010), doi:10.1016/j.jlap.2010.03.005
13. Manca, V., Marchetti, L.: Log-Gain Stoichiometic Stepwise regression for MP systems. IJFCS (to appear, 2010)
14. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
15. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
16. Pearson, K.: Notes on the History of Correlation. Biometrika 13(1), 25–45 (1920)

Modelling Spatial Heterogeneity and Macromolecular Crowding with Membrane Systems

Ettore Mosca[1], Paolo Cazzaniga[2], Dario Pescini[2], Giancarlo Mauri[2], and Luciano Milanesi[1]

[1] Institute for Biomedical Technologies, National Research Council, Via Fratelli Cervi, 20090 Segrate (MI), Italy
{ettore.mosca,luciano.milanesi}@itb.cnr.it
[2] Department of Informatics, Systems and Communications, University of Milan-Bicocca, Viale Sarca 336, 20126 Milan, Italy
{cazzaniga,pescini,mauri}@disco.unimib.it

Abstract. In biological processes, intrinsic noise, spatial heterogeneity and molecular crowding deeply affect the system dynamics. The classic stochastic methods lack of the necessary features needed for the description of these phenomena. Membrane systems are a suitable framework to embed these characteristics; in particular, the variants of τ-DPP and Sτ-DPP allow the modelling and stochastic simulations of multi-volume biochemical systems, in which diffusion and size of volumes and chemicals are taken into account improving the description of these biological systems. In this paper we show, by means of two models of reaction-diffusion and crowded systems, the correctness and accuracy of our simulation methods.

1 Introduction

Membrane systems [25] (also called P systems) have been recently exploited for the modelling of biological systems and for the investigation of their dynamical properties (we refer the reader to [28] for an updated bibliography). There are several important features that makes P systems suitable for the modelling of biological systems: a membrane structure is used to describe a compartmentalized environment, in which membranes are organized according to a specified hierarchy. Inside different membranes, different sets of objects (e.g. molecular species) can be defined along with sets of multiset rewriting rules (e.g. chemical reactions) to describe the evolution of the system.

On the other hand, there are some characteristics of the basic definition of P systems which are not adequate for the description of the biological reality. Among others, the maximal parallelism of the rules application, which consists in the application of all rules by consuming all objects present in the system, or the non deterministic selection of concurrent rules, are not suitable to the description of the stochasticity usually present in biological systems where some molecular

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 285–304, 2010.

species occur in small quantities. In order to achieve a better description of biological systems, the variant of dynamical probabilistic P systems (DPPs) [27] has been introduced. In DPPs, the maximal parallelism has been mitigated by assigning probabilities to the rules, and these values vary according to the system state. By exploiting these values, it is possible to provide a description of the system's dynamics, that is, DPPs allow to reproduce the stochastic variations of the elements (i.e. chemical species) occurring in the system. However, this description is only *qualitative*, in the sense that an effective (physical) time streamline cannot be directly associated to the evolution steps of the system.

τ-DPP has been introduced to overcome the limitations of DPPs, providing a *quantitative* description of a system dynamics, by extending the single-volume algorithm of tau-leaping [8].

In order to correctly describe the behaviour of a system, τ-DPP runs in parallel inside each volume. A modified version of the tau-leaping procedure presented in [8] is exploited to compute the length of the step τ. In this novel version of the simulation algorithm, the least value for the time increment, among those computed inside each volume, is used to sample the number of reactions to execute (as in the original tau-leaping algorithm). Thanks to this "common" time increment, shared by all volumes, the simulation is synchronized at each step, allowing the correct passage of the molecules involved in communication rules.

A novel variant of τ-DPP, presented in [11], has been introduced to consider the size of volumes and objects involved in a system, in order to better describe systems where the "space" play an important role in the dynamics, such as crowded systems. This variant of our multi-volume stochastic simulation algorithm, called Sτ-DPP, is based on the same modelling framework of τ-DPP, and it exploits the same strategy for the simulation of the system's behaviour.

τ-DPP and Sτ-DPP algorithms can be used in the modelling and simulation of reaction–diffusion (RD) systems and crowded environments. RD systems are mathematical models used to describe those chemical systems for which the spatial distribution of chemicals influence the overall dynamics. The standard methods used to describe such systems are based on partial differential equations; however, when the intrinsic fluctuations of the chemical system play a major role in the dynamic, as in the case of many systems of interest for biology, a stochastic approach is more suitable. The intracellular environment is considered crowded since it is characterised by the presence of high concentrations of soluble and insoluble macromolecules; therefore, the classic approaches which consider molecules as points (without specifying their size), are no longer adequate. Under crowded conditions, the rate of some cellular processes can be increased or decreased, according to the "free space" in the system. By using Sτ-DPP, it is possible to consider the size of reaction volumes and chemicals, achieving a correct description of a crowded system by computing the reactions probability according to the free space occurring in a volume.

In this paper we show the simulation of the heat equation by means of τ-DPP, proving its correctness in the simulation of diffusive processes. To this aim, we

compare the results of the stochastic simulations with the exact solution of the heat equation. Afterwards, we present the results of the simulation of a crowded environment, showing how macromolecules affect the reactions probability and the overall system dynamics.

The paper is organised as follows. In the next section we give a description of spatial heterogeneity and macromolecular crowding in living cells, and we briefly present the classic computational approaches used in the description of these phenomena; in Section 3, we describe the stochastic simulation algorithms we have developed for the modelling and description of multi-volume systems. In order to validate our approaches for the simulation of RD systems and crowded environments, we present in Section 4 the results obtained from the simulation of the heat equation, and in Section 5 the simulation of a biological system with macromolecules that induce crowding effects. We conclude with some remarks about the results of the presented systems and with some possible future extensions.

2 Spatial Heterogeneity and Macromolecular Crowding in Living Cells

Living cells are very far from the homogeneous and diluted compartment that is often used for their modelling. These requirements can be considered satisfied in many cases without taking them explicitly into account; however, there are several processes in which the effects of spatial heterogeneity (due to diffusive processes) and crowding (caused by the presence of macromolecules) must be considered in order to capture the correct system dynamics.

2.1 Reaction-Diffusion Systems

RD systems are mathematical models used to describe those chemical systems for which the spatial distribution of chemicals influences the overall dynamics. The standard approach exploits a continuous time and space domain description of the systems, such as the partial differential equations, where the mass transport, the chemical kinetics and the conservation laws, together with the boundary conditions, are embedded within the same set of equations that can be solved analytically or numerically.

When the intrinsic fluctuations of the chemical system play a major role in the dynamics, as is the case for many systems of interest for Biology, a master equation approach is more suitable [4,17]. The chemical master equation formulation adopts a mechanistic perspective on the chemical system describing it as a sequence of collision events among molecules. Each of these scattering events can lead either to a new compound (reactive collision) or to an elastic scattering (diffusive collision) which does not alter the chemical species distributions but only the particles speed and direction. Which of the two collisions pathways will be followed by each scattering event is determined by the energy involved in the process: if this energy exceed the Arrhenius threshold (activation energy)

then the two molecules will react to form the new compound. According to this scheme it is possible to separate the RD process into a free flight phase followed by the interaction phase. The resulting dynamics is the superposition of a brownian motion (random walk) with an interaction/reaction process. The existence of an activation energy imposes that the diffusive events are the most probable ones, if the environment in which the reactions take place is homogeneous, this picture corresponds to a well stirred reactor and the dynamics can be tracked by means of a stochastic simulation algorithm such as the Gillespie's one [17].

A natural extension of the master equation approach to heterogeneous (space) systems consists of dividing the original volume V into smaller sub-volumes V_v, each one with a characteristic length h ($V_v = h^d$, being $d \in \{1, 2, 3\}$ the spatial dimensions), such that each of these sub-volumes can be considered homogeneous. Moreover, it is possible to define a mean jump frequency $\widetilde{D}_{i,v}$ [4] for each chemical species S_i in the sub-volume V_v, in order to connect the microscopic description of the master equation with the macroscopic Fick's diffusion coefficient D_i

$$\widetilde{D}_{i,v} = \frac{2d}{h_v^2} D_i, \tag{1}$$

and to verify if the well stirred condition still holds. The latter requirements is equivalent to impose in each sub-volume that the diffusion time $\tau_D \approxeq \frac{h^2}{2dD}$ is much smaller then the reaction waiting time τ_R [5], condition that should be also granted for the molecules diffusing across the sub-volumes. This observation allows to define the expression for the stochastic kinetic constant associated to the "diffusive" reactions

$$c_D = \frac{D}{h^2} \tag{2}$$

that mimic the molecules movement from one sub-volume to another.

2.2 Macromolecular Crowding

The intracellular environment is characterised by the presence of high concentrations of soluble and insoluble macromolecules [16,23,34]. This medium is termed "crowded", "confined" or "volume-occupied", rather than "concentrated", because single molecular species may occur at low concentrations, but all species taken together occupy a considerable fraction of the total volume [24].

The term "macromolecular crowding" refers to the non-specific influence of steric repulsions (i.e., a consequence of the mutual impenetrability of molecules due to the Pauli exclusion principle) on molecular processes that occur in highly volume-occupied media [29].

Due to macromolecular crowding, biochemical, biophysical, and physiological processes in living cells may be quite different from those under idealized conditions [33], and order-of-magnitude effects of crowding have been demonstrated by both experimental and theoretical works on a broad range of processes [29]. All these effects are related to variations occurring in macromolecular thermodynamics activities [33] and diffusion [15].

To understand whether a crowded medium will increase or decrease the rate of a process, it is important to take into account the changes induced by the process itself on the available volume inside the system. Among others, binding of macromolecules to one another, folding of proteins and nucleic-acid chains into more compact shapes, the formation of aggregates, are all processes stimulated in crowding conditions due to the induced net increase of the available volume [15].

The other main effect of macromolecular crowding is related to anomalous diffusion. In crowded media, the mean squared displacement, $\langle r^2 \rangle$, of a solute particle in three dimensions is related to the diffusion coefficient D, but it is no longer linearly proportional to the time t:

$$\langle r^2 \rangle = 6Dt^{\alpha} \tag{3}$$

If $\alpha < 1$, the diffusion is called *anomalous subdiffusion*; on the other hand, if $\alpha > 1$ the diffusion is called *anomalous superdiffusion*; if $\alpha = 1$ the diffusion is normal. Crowding can reduce the rate of diffusion (according to the size of the diffusing molecule and to the degree of volume occupancy) and can lead to anomalous diffusion [3]. Large reductions in solute diffusion are probably indicators of interactions between the solute and cellular components, such as membranes [13]. Therefore, the rates of diffusion-controlled biochemical processes – mainly affected by the diffusion of the reactants – will be reduced in crowded media. The decrease in the diffusion rates due to crowding may also lead to complex phenomena like fractal kinetics (anomalous reaction orders and time-dependent reaction rate coefficients [21]) and spatial segregation of molecules [6]. In the latter case, as a consequence of the increased probability of recollision, crowding will determine the increase of the reaction rate of processes characterised by a low reaction probability [20].

2.3 Classic Computational Approaches

Computational approaches aimed at studying spatially heterogeneous systems and molecular crowding have to deal with the tabulation of spatial position of particles as a function of time. Several computational frameworks can be used to analyse such kind of systems [32]; we report hereafter the classic and most used methods.

Molecular dynamics (MD) simulations provide detailed trajectories, but they are computationally too expensive for simulating systems formed by a large number of atoms or with time scales above μs. MD has only been used in problems involving time-scales of ns and space-scales of tens of nm.

Brownian dynamics (BD) is a particle-based stochastic approach used to describe the time and space motion of molecules. Time and space are continuous, and noise is modelled by means of the Langevin equation. Crowded media can be explicitly described since it is possible to represent crowder molecules. However, as the number of particle collisions increases, the BD simulations demands a very high computational cost. Examples of methodologies based on BD are Green's function reaction dynamics algorithm [35], Smoldyn [2] and MCell [31].

Partial differential equations (PDEs) are a continuous and deterministic approach. PDEs represent the classical method used to model RD systems. Each equation relates the time variation of a species concentration to its space variation and to the other species concentration. Crowding effects can be implicitly represented acting on diffusion coefficients (e.g., by lowering their values) and kinetic constants (e.g., by increasing their values). PDEs are usually solved using numerical methods (only in a few case the analytical solution is available); moreover, as the time-step and the sub-volume size (the space domain is usually divided in a number of elements) are reduced, the solutions becomes more accurate while the computational effort increases.

Cellular automata (CA) consist of a grid of cells (in any number of dimensions), each cell has a finite set of states, and it evolves according to the neighbours state. CA can be used to simulate RD systems at both microscopic and mesoscopic scales, depending on the number of molecules associated with each cell of the lattice. Crowding can be explicitly represented by considering crowder molecules or fixed barriers. For instance, people of the CyberCell project modelled a virtual cell membrane using discrete automata [7].

Lastly, spatial approaches based on Gillespie's method extend the stochastic simulation algorithm [18] in order to represent an RD system as a set of a well-stirred chemical reactors that communicate particles. The next sub-volume method [14], spatial τ-leaping [30] and the method described in [5] are algorithms that follow this approach; MesoRD [19] and SmartCell [1] are popular simulators. As molecules are considered point particles, these tools cannot be utilised to describe molecular crowding.

3 Multi-volume Stochastic Simulation Algorithms Based on P Systems

The standard algorithms for the simulation of biochemical systems (see, for instance, the stochastic simulation algorithm (SSA) [18]) have been developed for the description of the exact behaviour of systems enclosed in a single volume. Recently, novel approaches have been introduced to simulate spatial heterogeneity, as in the next sub-volume method [14]. By using these methods, the volume of a system is divided into a number of separated sub-volumes, whose size is small enough to satisfy the requirements of the SSA, so that the probabilities of the reactions and the diffusive events occurring inside each sub-volume can be properly described.

A limitation of these stochastic methods consists in the fact that the size of chemicals and of the volumes in which reactions take place is not considered during the simulation of the system dynamics. To overcome the issues related to spatiality and size of chemicals, we have recently introduced two multi-volume stochastic algorithms called τ-DPP [9] and Sτ-DPP [11], which will be presented in the next subsections. These methods combine a variant of P systems called *dynamical probabilistic P systems* [27] with the simulation method of tau-leaping [8]: the first one is suitable for the description of chemical, biological and ecological systems, and can be easily applied to the modelling and simulation of

reaction-diffusion systems; the second one can be used for the simulation of crowded systems, since the size of chemicals and volumes is taken into account during the description of the system behaviour.

There exist different simulation methods based on P systems: among others we recall here the multicompartmental Gillespie's algorithm [26]. This algorithm is used to simulate systems composed by many volumes (also called compartments), exploiting the Gillespie's procedure. In particular, the direct method is used for the computation of the time τ and the index of the next reaction to execute within each compartment. This information is stored in a list which is updated at each iteration, modifying the values related to the compartments affected by the executed reaction. This strategy is very similar to that of the next subvolume method [14], with the difference that the multicompartmental Gillespie's algorithm does not use a particular data structure such as a heap or an indexed queue to efficiently handle compartments information.

In the description of biochemical systems modelled by means of this method, the so called *boundary rules* are exploited. Such kind of reactions are used to capture the features of the communication and the transformation of molecules. In particular, the authors consider special cases of boundary rules which involve molecules occurring in different compartments, though it is not very clear how Gillespie's theory for single volume systems can be used to describe the propensity functions of reactions that are simultaneously active in two compartments, nor in which compartment this information is used to compute the value of τ.

τ-DPP

τ-DPP [11] is a computational method which can be used to describe and perform stochastic simulations of complex biological or chemical systems. The "complexity" of the systems that can be managed by means of τ-DPP, resides both in the number of the (chemical) reactions and of the species involved, and in the topological structure of the system, that can be composed by many volumes. For instance, cellular pathways involving several spatial compartments (as the extracellular ambient, the cytoplasm, the nucleus, etc.), or multicellular systems like bacterial colonies, or multi-patched ecological systems as metapopulations, are all examples of complex systems that could be investigated with τ-DPP.

The correct behaviour of the whole system is achieved by letting all volumes evolve in parallel, and by using the following strategy for the choice of time increments. At each iteration of τ-DPP, we consider the current state of each volume (determined by the current number of molecules), and we calculate a time increment independently in each volume (according to the standard tau-leaping algorithm [8]). Then, the smallest time increment is selected and used to evaluate the next-step evolution of the entire system. Since all volumes *locally* evolve according to the same time increment, τ-DPP is able to correctly work out the *global* dynamics of the system. Moreover, by adopting this procedure, the simulated evolutions of all volumes get naturally *synchronized* at the end of each iterative step. The synchronization is also necessary – and exploited

together with a parallel update of all volumes – to manage the communication of molecules among volumes (i.e., diffusive events), whenever prescribed by specific (communication) rules.

The system is defined by means of a set of N volumes organised according to the hierarchy specified by the membrane structure. The state of the whole system is characterised by all multisets M_v occurring inside each volume V_v $(1 \leq v \leq N)$.

Inside the volumes, the sets of rules $R_1, \ldots, R_N$ are defined along with the sets of stochastic constants $C_1, \ldots, C_N$.

Each volume V_v can contain two different kinds of rules, termed *internal* and *communication* rules. An internal rule describes the modification, or evolution, of the objects inside the single volume where it is applied, while a communication rule sends the objects from the volume where it is applied to an adjacent volume (possibly modifying the form of these objects during the communication step).

More precisely, internal rules have the general form $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_m S_m \rightarrow \beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_m S_m$, where $S_1, \ldots, S_m$ belong to the set of distinct object types $\mathcal{S}$, and $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \in \mathbb{N}$. For instance, $S_1, \ldots, S_m$ can correspond to molecular species, and, in this case, $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m$ represent stoichiometric coefficients. The objects appearing in the left-hand side of the rule are called *reagents*, while the objects on the right-hand side are called *products*. Note that, usually, we will consider the case where (at most) three objects appear in the reagents group. The rational behind this is that we require biochemical reactions to be (at most) of the third-order, since the simultaneous collision and chemical interaction of more than three molecules at a time, has a probability to occur close to zero in real biochemical systems. Moreover, the interaction among more than three molecules can be modelled by using a set of successive reactions with lower order. In what follows, we will refer to rules or reactions without distinction.

When dealing with communication rules inside a volume, besides defining the sets of reagents and products, it is necessary to specify the target volume where the products of this rule will be sent[1]. Formally, a communication rule has the form[2] $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_m S_m \rightarrow (\beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_m S_m, tar)$, where $S_1, \ldots, S_m \in \mathcal{S}$ are distinct object types, $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \in \mathbb{N}$, and tar represents the volume where the products of the reaction diffuse.

A complete an extensive description of the τ-DPP algorithm and some applications can be found in [9,10].

Sτ-DPP

Sτ-DPP [11] is obtained combining the structure definition of tissue P systems [22] with the simulation strategy used in τ-DPP [9]. Here, nodes are arranged

[1] This definition can be easily extended in order to assign a different target volume to each object appearing in the set of products.

[2] The condition that at most three objects appear as reagents is usually required also for communication rules.

in a tissue–like fashion, but each of them can have a complex internal hierarchy, organised in a tree–like structure. Moreover, in this new variant we consider sizes for both membranes and objects, and the rules defined inside each membrane will be enabled only in the case there is sufficient free space in the membrane where the rule is applied, for instance, to "create" new objects or to receive objects from other volumes. The size considered here can be used in the modelling and simulation of biochemical systems where diffusive processes play an important role, and it is necessary to avoid the unlimited accumulation of objects in a region of finite size.

In order to correctly describe the hierarchy of complex nodes of the system we first need a graph representing the topology of the membranes. In particular, this graph can have undirected edges to indicate that two membranes are placed at the same hierarchical level (as in the standard definition of tP systems [22]). On the other hand, directed edges of the graph are used to denote that the "source" membrane contains the "target" membrane.

A second directed graph is needed to represent the communication channels among membranes. Clearly, the arrows of the edges indicate the direction of the (permitted) flow of objects between different compartments. Note that, this communication graph can contain edges that are not indicated in the graph which describes the membrane structure. The meaning of these particular edges is to represent communication channels that connect non adjacent membranes. Using these arcs, it is then possible to create privileged pathways of communication between membranes.

The features of Sτ-DPP can be exploited to represent (among other real life systems) reaction–diffusion systems [12]. In this case, the membrane structure can be used to represent a reaction volume as the composition of a number of finite size sub-volumes, and the communication graph will describe the diffusion directions through the system.

With respect to the definition of τ-DPP, when using this simulation strategy we need to specify additional information about the system under investigation. First, we need to provide the sizes of the volumes composing the system and of the molecular species occurring within them. Second, we have to compute the initial free space of each volume.

Given the internal state M_v of a membrane V_v together with size $s_{S_1}, \ldots, s_{S_m}$ of the species, and the size $s_{V_1}, \ldots, s_{V_N}$ of the volumes, it is possible to define the *occupied volume* in V_v as:

$$O(V_v) = \sum_{i=1}^{m} (m_i \cdot s_{S_i}) + \sum_{V_l \in a_{\mathcal{T}}(V_v)} s_{V_l}$$

where m_i represents the amount of the species S_i and $a_{\mathcal{T}}(V_v)$ is the set of volumes contained within the volume V_v. Hence, it is possible to define the value of the *free space* in V_v as:

$$F(V_v) = s_{V_v} - O(V_v)$$

Internal and communication rules are defined as in τ-DPP, but, at each rule execution (internal or communication), apart from updating the molecular

amounts of the species involved in the reaction, also the free space value has to be updated. The update operation adds to the free space value the "volume" of the objects consumed or sent by the rule and subtracts the "volume" of the objects produced by the rule or received from other membranes. In particular, after the execution of an internal rule, the free space in V_v is updated as $F(V_v) = F(V_v) - \sum_{i=1}^{m} (\beta_i - \alpha_i) \cdot s_{S_i}$. On the contrary, when a communication rule is applied, we need to update the free space of V_v (i.e., the membrane where the reaction is applied) as $F(V_v) = F(V_v) + \sum_{i=1}^{m} \alpha_i \cdot s_{S_i}$ and the free space of each target volume V_{tgt_k} indicated by the rule: $F(V_{tgt_k}) = F(V_{tgt_k}) - \sum_{i=1}^{m} \beta_{i,k} s_{S_i}$.

At each iteration of the algorithm, in order to obtain a correct description of the system's dynamics, we need to check if a rule r (internal or communication) is applicable. A complete description of Sτ-DPP can be found in [11]; some additional details about the implementation of the algorithms will be provided in the next section.

Handling Diffusive Events and Crowding in τ-DPP and Sτ-DPP Algorithms

The iterations of the simulation algorithms described in the previous sections are composed of three main stages: (i) computation of the reactions probability; (ii) calculation of the τ value; (iii) selection of the reactions to execute and check of the system state consistency. During these stages we have to keep into account some details in order to correctly describe diffusive events and the effect due to crowded media, as described in what follows.

In the first stage, given the system state $\mathbf{x}$, the probability $a(\mathbf{x})$ of a rule application (i.e. the propensity function) is generally computed as follows: $a(\mathbf{x}) = c \cdot h$, where c is the stochastic constant associated to the rule and h is a combinatorial function depending on the left-hand side of the rule [18]. This definition is used in τ-DPP to compute the propensity functions of each volume of the system. Note that, this operation is performed independently in each volume; hence, the propensity function of the reactions occurring inside a volume V_v depends only on its current state (defined as M_v).

For what concerns the Sτ-DPP algorithm, the propensity functions of the internal reactions are computed by also considering the value of the free space of the current volume. So doing, we can correctly simulate crowded systems: we suppose that while first order reactions (e.g. $a \rightarrow b$) are not affected by the value of the free space, in the case of reactions of higher orders, the lack of free space enhances the reaction probability. Therefore, the propensity functions of second and third order reactions are computed as follows:

$$a(\mathbf{x}) = \frac{c \cdot h}{F(V)}. \tag{4}$$

On the other hand, communication rules representing diffusive events are not influenced by the free space left in the volume; therefore, their propensity functions are computed as in the standard procedure.

During the second phase of the algorithms, inside each volume of the system, a candidate length τ of a step is obtained by using the following equation:

$$\tau = \min_{i \in \mathcal{S}} \left\{ \frac{max\{\varepsilon m_i / g_i, 1\}}{|\mu_i(\mathbf{x})|}, \frac{max\{\varepsilon m_i / g_i, 1\}^2}{\sigma_i^2(\mathbf{x})} \right\},$$

where g_i is a value depending on the highest order of reaction in which a species $i \in \mathcal{S}$ is involved and ε is an error control parameter, while $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$ are calculated as described below (according to the definition presented in [8]):

$$\mu_i(\mathbf{x}) = \sum_{j \in R_{ncr} \cap R_{int}} (v_{i,j}\ a_j(\mathbf{x})) + \sum_{j \in R_{ncr} \cap R_{comm}} (-lhs_{i,j}\ a_j(\mathbf{x}))\,, \quad \forall i \in \mathcal{S},$$

$$\sigma_i^2(\mathbf{x}) = \sum_{j \in R_{ncr} \cap R_{int}} \left(v_{i,j}^2\ a_j(\mathbf{x})\right) + \sum_{j \in R_{ncr} \cap R_{comm}} \left(-lhs_{i,j}^2\ a_j(\mathbf{x})\right), \quad \forall i \in \mathcal{S},$$

where j belongs to the set R of reactions of the considered volume, the restriction on the set of noncritical reactions R_{ncr} is present, due to the conditions of the modified non-negative Poisson tau-leaping [8], while R_{int} represents the set of internal rules and R_{comm} the set of communication rules. During the computation of μ_i and σ_i^2, we consider the variation of the species i due to the reaction j (specified by the value $v_{i,j}$), for what concerns internal rules. On the other hand, we only consider the variation of the species i described by the left-hand side of a communication rule j ($lhs_{i,j}$), since the current volume is affected only by these variations.

There exists different approaches in which communication rules (i.e. diffusive processes) are considered as special events and their probabilities are computed by using a deterministic formulation, and during the calculation of μ_i and σ_i^2 besides the changes of the species due to the rule, also the contribution of the neighbourhood volumes is taken into consideration (see, for instance, [30]).

In the last part of the algorithms, the set of reactions to be executed inside each volume is selected, and before the system update, the applicability of this set has to be verified in order to obtain a consistent system state.

Both τ-DPP and Sτ-DPP select the number of occurrences of each reaction j (inside each volume) by sampling a random number from a Poisson distribution having mean and variance equal to $a_j(\mathbf{x})\tau$. Afterwards, the applicability of the set of selected reactions is verified: both algorithms check if the system state resulting from the execution of the reactions contains negative values for the amount of some molecular species. Moreover, in the case of Sτ-DPP, the set of reactions selected in a volume is applicable only if there is enough free space after the rules application [11].

As stated above, a rule can be executed only if the free space of the volume, after the rule application, is greater or equal to zero. The rule applicability is computed differently for internal and communication rules. Given an internal rule occurring inside volume V_v, we need to check if, after the rule execution, $F(V_v) \geq 0$. For what concerns a communication rule r, we also need

to check all the free space of all the target volumes indicated by the rule: $\forall\ tgt_l$ of r, $F(V_{tgt_l}) \geq 0$, where the values β_j are the stoichiometric coefficients of the molecular species associated with V_{tgt_l}.

Note that, using a strategy based on the tau-leaping algorithm to describe the behaviour of the system, at each iteration step a certain number of rules is applied in parallel. Hence, the applicability of the entire set of selected rules has to be verified. This operation is realised by computing the free values of each volume V_v considering the contribution of all the selected rules; if the values of $F(V_v)$ is greater or equal to zero (for each volume), then the execution is allowed.

If any of these requirements is not satisfied, then the value of τ is reduced by half and a new set of reactions is selected (this strategy has been proposed in [8]). On the other hand, the iteration of the τ-DPP and Sτ-DPP algorithms proceeds by updating the system state and the simulation time; and the simulation finishes if a termination criterion is reached.

4 Validation of the Diffusion Implemented with τ-DPP

In this section we present some results obtained from simulations performed by using τ-DPP in order to verify if diffusion is correctly handled by our simulation algorithm. Berstein [5] showed that it is possible to simulate mesoscopic RD systems using the Gillespie's algorithm comparing the simulations with the solution of diffusion equations. We adopted the same strategy in order to show that τ-DPP can be used to reproduce diffusion introducing a reasonably small error.

4.1 A Popular Diffusion Equation: The Heat Equation

The heat equation is a partial differential equation which describes the heat distribution in a region during time, and it is a special case of diffusion equation where the diffusion coefficient D is constant in time and space:

$$\frac{\partial u(\overrightarrow{x}, t)}{\partial t} = D\Delta u(\overrightarrow{x}, t) \tag{5}$$

where $u(\overrightarrow{x}, t)$ is the density of the diffusive material in $\overrightarrow{x}$ at time t, and $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ is the Laplacian operator.

To test the accuracy of τ-DPP in reproducing diffusion, we studied the unidimensional diffusion of the molecule S in the region $\Omega \subset \mathbb{R}$:

$$\frac{\partial [S](x,t)}{\partial t} = D\frac{\partial^2 [S](x,t)}{\partial x^2},\ \forall x \in \Omega \tag{6}$$

where $[S](x,t)$ indicates the concentration of molecule S in position x at time t. In particular, we considered the region $\Omega = [0,1]$ and the Neumann boundary conditions:

$$\frac{\partial [S](0,t)}{\partial x} = \frac{\partial [S](1,t)}{\partial x} = 0, \tag{7}$$

indicating that the flux from outside into Ω is null. Considering $D = 1$, an exact solution in the region Ω satisfying Eq. 7 is:

$$[\mathrm{S}](x,t) = S^{\Omega}[1 + \tfrac{1}{2}\mathrm{e}^{-\pi^2\gamma^2 t}\cos(\gamma\pi x)] \tag{8}$$

where S^{Ω} is the total number of S molecules inside the system and γ is a non negative integer. In all the cases that we will discuss in the next section we have considered $\gamma = 3$ and $S^{\Omega} = 500$.

4.2 Comparison between τ-DPP and the Heat Equation

In order to compare the simulations performed by using τ-DPP with the continuous exact solution of the heat equation (Eq. 8) at a given time instant, we dived the region Ω into N smaller adjacent regions V_v such that $\Omega = \bigoplus_v V_v$, and $V_v = \Omega/N$.

A series of issues have to be handled to realize a meaningful comparison. First, since τ-DPP algorithm is stochastic, it is crucial to consider a high number of simulations in order to obtain a significant comparison with the heat equation. We accomplished this task averaging the results of a sufficiently high number G of simulations. Note that, in general, this average is not representative of the final state of a system, like in the case of multistable systems, in which averaging may lead to fictitious states. However, when the average solution converges to the system state – as in the case we are considering here – the deviations from the exact solution can be considered as a type of sampling error and the average solution is a good representative of the system state.

Second, since τ-DPP simulator works with molecules, rather than molecule concentration, we must calculate the initial distribution of S molecules within V_v, i.e., $S_1, \ldots, S_N$, from the solution of Eq. 8 at time $t = 0$, in order to use this distribution as input for τ-DPP. Moreover, we must calculate τ-DPP predicted concentrations, $[S^*]_1, \ldots, [S^*]_N$, from the distribution of S molecules over the sub-volumes at a particular time point. These conversions have been defined according to the following relation between concentration $[S]$ and molecule number S:

$$[S_v] = \frac{S_v}{V_v},\ \forall v \tag{9}$$

Note that the solution of Eq. 8 must be calculated using the appropriate vector $\boldsymbol{x} = (x_1, \ldots, x_v, \ldots, x_N)$, whose members are located at the middle of each volume s_{V_v}:

$$x_v = \frac{V_v}{2} + (v-1)V_v, v = 1, \ldots, N \tag{10}$$

Third, since the time increments τ are randomly generated, it is very unlikely that the simulator will output a numerical solution exactly at a specific time point t. Therefore, the molecule distribution computed by τ-DPP at a particular time point t has been calculated as a linear interpolation of the two numerical solutions at $t_1 < t$ and $t_2 > t$, where t_1 and t_2 are, respectively, the points

computed by the τ-DPP that immediately precede and follow t. An example of comparison between the heat equation and the τ-DPP simulations is shown in Fig. 1, where it is possible to observe the high closeness between the simulations and exact solution.

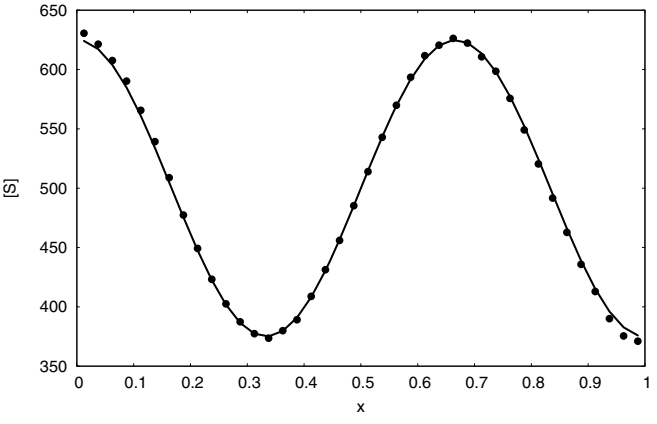


Fig. 1. The heat equation exact solution (line) and τ-DPP average results (dots) at $t = 0.0078034$, $S^{\Omega} = 500$, $G = 10000$, $\gamma = 3$, $s_{V_v} = 0.025$

Quantitatively, the quality of the τ-DPP simulation $e = 1, \ldots, G$ has been assessed considering, as in [5], the error due to the difference between the exact solutions $[S_v]$ and the concentrations computed using the numerical results of τ-DPP $[S^*_{v,e}]$:

$$\epsilon_v = \frac{1}{G} \sum_{e=1}^{G} \left(1 - \frac{[S^*_{v,e}]}{[S_v]} \right) \tag{11}$$

in the volume v, considering a pool of G simulations ran with the same settings. Note that $\epsilon_v \to 0$ as $[S^*_{v,e}] \to [S_v]$ (obviously) and in the case in which the distribution of $[S^*_{v,e}]$ is symmetric with respect to the exact value $[S_v]$. The errors ϵ_v have been averaged considering all the elements of Ω:

$$\epsilon = \frac{1}{N} \sum_{v=1}^{N} \mid \epsilon_v \mid \tag{12}$$

We studied the relation of ϵ with the number of simulations $10 \leq G \leq 10000$ and the number of volumes $10 \leq N \leq 40$ used to discretise the spatial region Ω.

As the number of simulations increases the sampling error decreases (as shown in Fig. 2). In particular, we observed a decrease of one order of magnitude passing from 10 to 10^4 simulations in all cases with exception of $s_{V_v} = 0.1$ ($N = 10$), where the decrease has been lower. Note that as $G \to 0$ the lowest error is

associated to settings with a higher s_{V_v}, while as $G \to \infty$ the lowest error is associated to lower s_{V_v}. This can be attributed to the noise: as $s_{V_v} \to 0$, the volumes will contain a lower number of molecules (high noise); hence, a higher number of simulations is required to eliminate noise. This phenomenon has been particularly evident in the study presented here due to the relatively low quantity of molecules used, $S^{\Omega} = 500$, with respect to the number of volumes for the discretisation of Ω, $10 \leq N \leq 40$: we passed from $S_v \in [10, 10^2]$ when $s_{V_v} = 0.1$ $(N = 10)$ to $S_v \in [1, 10]$ when $s_{V_v} = 0.025$, $(N = 40)$.

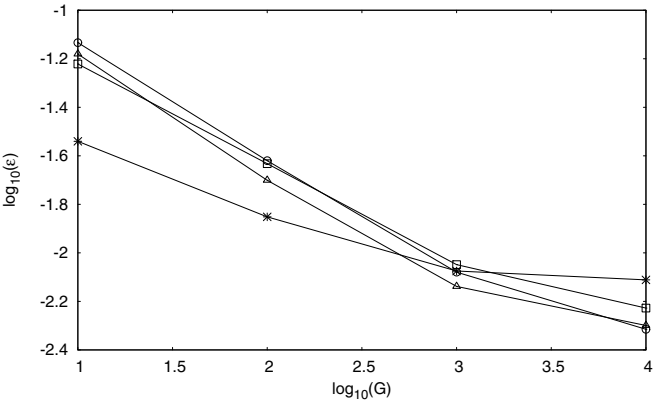


Fig. 2. Relation between the error ϵ and the number of simulations (G). $s_{V_v} = 0.1$ (*), $s_{V_v} = 0.05$ (□), $s_{V_v} = 0.03\overline{3}$ (△), $s_{V_v} = 0.025$ (○), $\gamma = 3$, $t = 0.0078034$.

Another source of error is associated with the spatial discretisation, i.e. with the number of membranes in which Ω is divided into. In order to reduce the contribution of the sampling error it is crucial to study the behaviour of the spatial discretisation error with a high G. This error decreases as $s_{V_v} \to 0$ (Fig. 3).

5 Macromolecular Crowding with Sτ-DPP

Sτ-DPP allows to model objects of arbitrary size that react and diffuse through a spatial region composed of membranes of arbitrary size; therefore, Sτ-DPP can be used to simulate crowded RD systems. In a previous work, we have shown that Sτ-DPP are able to reproduce systems in which crowding determines a slower motion of molecules [11].

In this section we show the effects of macromolecular crowding which increases a biochemical reaction rate due to the increase of reaction collision probabilities, that is in turn determined by the reduction of the free space induced by crowding. In particular, the reduction of the space in a volume determines the increase of the propensity functions of second and third order reactions, as described in Section 3.

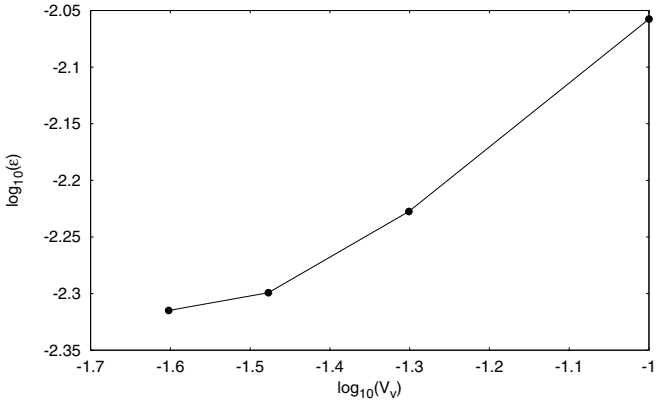


Fig. 3. Relation between the spatial discretisation error and the volumes size at time $t = 0.0078034$

We accomplished this task by modelling a crowded RD system composed by four species, A, B, C, Z, within the spatial region $\Omega \subset \mathbb{R}^2$, represented using a set of 81 membranes organised as a 9x9 bidimensional lattice, and two biochemical reactions:

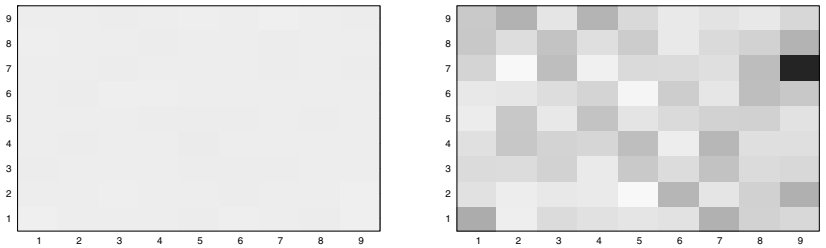
$$r_1 : \mathrm{A} + \mathrm{B} \xrightarrow{c_1} \mathrm{C} \tag{13}$$

$$r_2 : \mathrm{C} \xrightarrow{c_2} \mathrm{A} + \mathrm{B} \tag{14}$$

where $c_1 = 0.0001$ and $c_2 = 0.001$ are the stochastic constants. Stochastic constants for diffusion rules have been set three orders of magnitude higher than c_1, c_2, that is $c_{D_A} = c_{D_B} = c_{D_C} = 0.16$, in order to ensure $\tau_D \ll \tau_R$. The size of species Z is three orders of magnitudes bigger than that of the other species ($s_Z = 0.1$, $s_A = s_B = 0.0001$ and $s_C = 0.0002$) and it has been used to represent macromolecular crowding. The simulation has been done considering 100 molecules of A and 100 molecules of B in each membrane while molecule Z have been randomly distributed among the membranes; the number of Z has been chosen in order to occupy 0 (diluted media) or approximately $\frac{1}{3}$ (crowded media) of the total volume. Note that, while molecules A, B, C can diffuse to each first next neighbour, molecules Z do not diffuse.

For each membrane we determined the average value assumed by the propensity function associated with the reaction defined in Eq. 13 (i.e. production of C), during the simulation in the time interval $[0, 10]$.

In the diluted condition the reaction propensity assumes approximately the same value within all membranes, with mean value $\mu = 0.9237$ and standard deviation $\sigma = 0.001842$ (Fig. 4(a)). The distribution of the propensity functions values has been affected by the addiction of crowding molecules: in this condition we obtained mean $\mu = 1.451$ and standard deviation $\sigma = 0.498$ (Fig. 4(b)). The average reaction propensity values in the crowded media are up to 6 folds higher



(a) (b)

Fig. 4. Average propensity function values of reaction r_1 within the spatial domain in (a) diluted ($\mu = 0.9237, \sigma = 0.001842$) and (b) crowded conditions ($\mu = 1.451, \sigma = 0.498$)

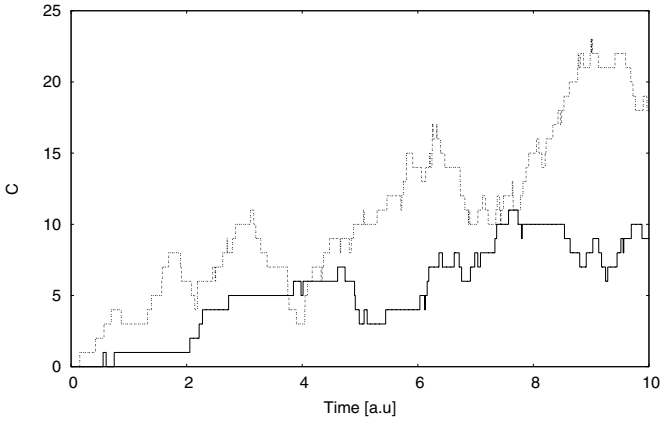


Fig. 5. Number of C molecules in the crowded volume 78 (row 7 and column 9 of Fig. 4) in diluted (solid line) and crowded (dashed line) conditions

then in the diluted case, and their homogeneity over the system is affected by the presence of highly reactive volumes in which the production of molecules C is faster, as shown in Fig. 5.

6 Conclusions

In this work we examined the application of two membrane systems variants, τ-DPP and Sτ-DPP, in order to model and simulate spatial heterogeneity and molecular crowding, two important characteristics of living cells.

To capture spatial heterogeneity at the molecular level, a spatial domain must be defined and the time evolution of possibly reacting molecules must be tracked in different locations of the domain. We showed that τ-DPP can reproduce the diffusion of molecules within a spatial region divided in a set of sub-volumes (membranes) that can move (communicate) objects, according to a defined topology.

We tested diffusion using the same strategy followed by Bernstein [5], i.e., we compared the τ-DPP simulations with an analytical solution of a PDE for the heat equation (a diffusion equation). Note that for most applications of real interest biological processes analytical solutions are hardly available.

The error that we reported is slightly greater than the one found in [5]. This is due to the fact that the τ-leaping algorithm – which stands at the basis of τ-DPP – generates an approximate dynamics with respect to the exact solution of the chemical master equation, whereas the Gillespie's algorithm used in [5] is exact. We think that this loss of accuracy (that can be a priori controlled) is well balanced by the increase in performance that enables simulations of more complex systems. The quantitative characterisation of this discrepancy is currently under investigations and it will be published in a future work.

Molecular crowding can be explicitly modelled by the Sτ-DPP variant, an extension of τ-DPP, in which - among other features - sizes are associated to objects and membranes. We have previously shown that Sτ-DPP captures the delay in the communication of objects between volumes due to crowded conditions [11]. In this contribution we showed that Sτ-DPP can also be used to reproduce the reaction rate increase observed in crowded media due to the increase of recollision probability determined by the reduction of the available free space in a volume. This effect has been captured by modifying the propensity function calculation of second and third order reactions, and it has been shown with an example concerning a crowded RD system in a bidimensional region.

Considering the results provided in this work, Sτ-DPP can be successfully used to model and simulate crowded RD systems. It is worthy to note that, the current version of Sτ-DPP has all the features to capture the major effects that a crowded medium determines over RD system dynamics, and we plan to do an extensive study of this topic in future. Moreover, an interesting direction of investigation, enabled by the possibility of arbitrarily defining the volumes size and the topology of their communication, consists in the use of Sτ-DPP to study RD systems dynamics in structured regions (as the cytoplasm of living cells).

Acknowledgements

This work has been supported by the NET2DRUG, EGEE-III, BBMRI, EDGE European projects and by the MIUR FIRB LITBIO (RBLA0332RH), ITALBIONET (RBPR05ZK2Z), BIOPOPGEN (RBIN064YAT), CNR-BIOINFORMATICS initiatives, FAR-09 "Modelli di calcolo naturale e applicazioni" Italian initiatives and "Dote Ricercatori": FSE, Regione Lombardia.

References

1. Ander, M., Beltrao, P., Ventura, B.D., Ferkinghoff-Borg, J., Foglierini, M., Kaplan, A., Lemerle, C., Toms-Oliveira, I., Serrano, L.: Smartcell, a framework to simulate cellular processes that combines stochastic approximation with diffusion and localisation: analysis of simple networks. Systems Biology 1(1), 129–138 (2004)

2. Andrews, S.S., Addy, N.J., Brent, R., Arkin, A.P.: Detailed simulations of cell biology with smoldyn 2.1. PLoS Computational Biology 6(3), e1000705 (2010)
3. Banks, D.S., Fradin, C.: Anomalous diffusion of proteins due to molecular crowding. Biophysical Journal 89(5), 2960–2971 (2005)
4. Baras, F., Mansour, M.: Reaction-diffusion master equation: A comparison with microscopic simulations. Physical Review E 54(6), 6139–6148 (1996)
5. Bernstein, D.: Simulating mesoscopic reaction-diffusion systems using the gillespie algorithm. Physical Review E 71(4 Pt 1), 041103 (2005)
6. Berry, H.: Monte carlo simulations of enzyme reactions in two dimensions: fractal kinetics and spatial segregation. Biophysical Journal 83(4), 1891–1901 (2002)
7. Broderick, G., Ru'aini, M., Chan, E., Ellison, M.J.: A life-like virtual cell membrane using discrete automata. In Silico Biology 5(2), 163–178 (2005)
8. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. Journal of Chemical Physics 124, 44109 (2006)
9. Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G.: Tau leaping stochastic simulation method in p systems. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 298–313. Springer, Heidelberg (2006)
10. Cazzaniga, P.: Stochastic algorithms for biochemical processes. Ph.D. thesis, Università degli Studi di Milano-Bicocca (2010)
11. Cazzaniga, P., Mauri, G., Milanesi, L., Mosca, E., Pescini, D.: A novel variant of tissue p systems for the modelling of biochemical systems. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 210–226. Springer, Heidelberg (2010)
12. De Wit, A.: Spatial patterns and spatiotemporal dynamics in chemical systems. Advances in Chemical Physics 109, 435–513 (1999)
13. Dix, J.A., Verkman, A.S.: Crowding effects on diffusion in solutions and cells. Annual Review of Biophysics 37, 247–263 (2008)
14. Elf, J., Ehrenberg, M.: Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. In: IEE Proceedings Systems Biology, vol. 1(2), pp. 230–236 (2004)
15. Ellis, R.J., Minton, A.P.: Cell biology: join the crowd. Nature 425(6953), 27–28 (2003)
16. Fulton, A.B.: How crowded is the cytoplasm? Cell 30(2), 345–347 (1982)
17. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics 22(4), 403–434 (1976)
18. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
19. Hattne, J., Fange, D., Elf, J.: Stochastic reaction-diffusion simulation with mesoRD. Bioinformatics 21(12), 2923–2924 (2005)
20. Kim, J.S., Yethiraj, A.: Effect of macromolecular crowding on reaction rates: a computational and theoretical study. Biophysical Journal 96(4), 1333–1340 (2009)
21. Kopelman, R.: Fractal reaction kinetics. Science 241(4873), 1620–1626 (1988)
22. Martín-Vide, C., Pazos, J., Păun, G., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue p systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 290–299. Springer, Heidelberg (2002)
23. Minton, A.P.: The effect of volume occupancy upon the thermodynamic activity of proteins: some biochemical consequences. Molecular and Cellular Biochemistry 55(2), 119–140 (1983)

24. Minton, A.P.: The influence of macromolecular crowding and macromolecular confinement on biochemical reactions in physiological media. Journal of Biological Chemistry 276(14), 10577–10580 (2001)
25. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
26. Pérez-Jiménez, M.J., Romero-Campero, F.J.: P Systems, a New Computational Modelling Tool for Systems Biology. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 176–197. Springer, Heidelberg (2006)
27. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. International Journal of Foundations of Computer Science 17(1), 183–204 (2006)
28. The P systems web page, http://ppage.psystems.eu
29. Rivas, G., Ferrone, F., Herzfeld, J.: Life in a crowded world. EMBO Reports 5(1), 23–27 (2004)
30. Rossinelli, D., Bayati, B., Koumoutsakos, P.: Accelerated stochastic and hybrid methods for spatial simulations of reaction-diffusion systems. Chemical Physics Letters 451(1-3), 136–140 (2008)
31. Stiles, J.R., Bartol, T.M.: Monte Carlo Methods for Simulating Realistic Synaptic Microphysiology Using MCell. In: Computational Neuroscience: Realistic Modeling for Experimentalists, pp. 87–127. CRC Press, Boca Raton (2001)
32. Takahashi, K., Arjunan, S.N.V., Tomita, M.: Space in systems biology of signaling pathways–towards intracellular molecular crowding in silico. FEBS Letters 579(8), 1783–1788 (2005)
33. Zimmerman, S.B., Minton, A.P.: Macromolecular crowding: biochemical, biophysical, and physiological consequences. Annual Review of Biophysics and Biomolecular Structure 22, 27–65 (1993)
34. Zimmerman, S.B., Trach, S.O.: Estimation of macromolecule concentrations and excluded volume effects for the cytoplasm of escherichia coli. Journal of Molecular Biology 222(3), 599–620 (1991)
35. van Zon, J.S., ten Wolde, P.R.: Green's-function reaction dynamics: a particle-based approach for simulating biochemical networks in time and space. Journal of Chemical Physics 123(23), 234910 (2005)

Randomized Gandy-Păun-Rozenberg Machines

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences,
Śniadeckich 8, P.O. Box 21, 00-956 Warsaw, Poland
adamo@impan.gov.pl

Abstract. An idea of a randomized Gandy–Păun–Rozenberg machine providing a certain abstract implementation of concurrent (parallel) randomized algorithms is introduced. A randomized Gandy–Păun–Rozenberg machine for solving 3-SAT problem in a polynomial time with the low error probability and with subexponential number of indecomposable processors is shown. This machine assembles a distributed system which then realizes a massively parallel computation.

1 Introduction

We propose and discuss an idea of a randomized Gandy–Păun–Rozenberg machine which is a randomized counterpart of a concept of a Gandy–Păun–Rozenberg machine introduced in [5] and recalled in Section 2 of the present paper, where one finds the connections with membrane computing.

In general randomized Gandy–Păun–Rozenberg machines, briefly called randomized G–P–R machines, are aimed to serve for an abstract implementation of concurrent (parallel) randomized algorithms or to provide some description of these algorithms.

Here one can describe informally a *randomized algorithm* as an algorithm which contains a possibly deterministic algorithm (test) performed for some randomly chosen input data in a possibly polynomial time, where there is known some estimation of error probability of the final result of the test; for a more formal approach see [6]. Randomized algorithms are used when for their tasks there are not known efficient deterministic algorithms.

In particular a goal of randomization of Gandy–Păun–Rozenberg machines, yielding randomized G–P–R machines, is to decrease the exponential expansion of the number of indecomposable processors which appear in computations of G–P–R machines constructed in the manner of [5] to solve NP complete problems in a polynomial time. A decreasing of exponential expansion of the number of indecomposable processors to some subexponential one is achieved with a loss of certainty of a final result which is reached with some error probability in a similar way as in the case of randomized algorithms, where a subexponential time of computations is achieved with a loss of certainty of a final result, cf. [4]. Randomized G–P–R machines are similar to Gandy–Păun–Rozenberg machines

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 305–324, 2010.

except the initial instantaneous descriptions of randomized G–P–R machines contain certain configurations chosen at random. These randomly chosen configurations steer the computations in such a way that subexponential number of indecomposable processors appear in the computations.

The subexponentiality is understood here in the following way. For the length k of input data the time or space complexity measures with respect to k of a given algorithm are called *subexponential* if they are bounded by a function $e^{f(k)}$ for some function $f(k)$ with $\lim_{k\to\infty} \frac{f(k)}{k} = 0$, see [4].

We present in Section 3 an explanatory example of a randomized G–P–R machine which solves 3-SAT problem in a polynomial time with low error probability and with subexponential number of indecomposable processors. The example shows that a parallelized randomization or randomized parallelization of computations is possible, which is explained in the conclusion of the paper.

The example also shows a theoretical (discrete-topological) possibility of a construction of a system, still in Gandy's mechanism frames[1] [2], whose computation process consists of two phases: a phase of assembly of a distributed system (possibly with randomly chosen input data), and then a phase of massively parallel computation realized by this distributed system.

2 Gandy-Păun-Rozenberg Machines; Examples

An idea of a Gandy–Păun–Rozenberg machine, briefly G–P–R machine, introduced in [5], is aimed to provide an answer to the question:

what is it an $\mathcal{X}$ possible machine?

for $\mathcal{X} \equiv$ set-theoretically, $\mathcal{X} \equiv$ discrete topologically, and $\mathcal{X} \equiv$ biologically inspired.

The G–P–R machines are the constructs which have common features with or are related to:

— Gandy's machines [2], [9],
— P systems due to Gheorghe Păun (cf. [8]),
— parallel rewriting systems of graphs investigated by Grzegorz Rozenberg himself with scientists cooperating with him, among others, in preparation and editing of multi-volume *Handbook of graph grammars and computing by graph transformation* [3].

The core of a G–P–R machine is a finite set of rewriting rules for certain finite directed labelled graphs, where these graphs are instantenous descriptions for the computation process realized by the machine.

The conflictless parallel (simultaneous) application of the rewriting rules of a G–P–R machine is realized in Gandy's machine mode (according to Local Causality Principle), where (local) maximality of "causal neighbourhoods" replaces (global) maximality of, e.g. conflictless set of evolution rules applied simultaneously to a membrane structure which appears during the evolution process

[1] Via the representation of G–P–R machines by Gandy machines given in [5].

generated by a P system. Therefore one can construct a Gandy's machine from a G–P–R machine in an immediate way, see [5].

The NP complete problems can be solved by G–P–R machines in a polynomial time (but with an exponential number of indecomposable processors), see [5], where one constructs a G–P–R machines solving SAT problem in a polynomial time in a similar way to (families of) P systems solving this problem also in a polynomial time (cf. Păun's pioneering paper [7]).

For all unexplained terms and notation of category theory and graph theory we refer the reader to Appendix.

Definition. A G–P–R *machine* $\mathcal{M}$ is determined by the following data:

— a finite set $\Sigma_{\mathcal{M}}$ of labels or symbols of $\mathcal{M}$,
— a skeletal set $\mathcal{S}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over Σ, which are called *instantenous descriptions* of $\mathcal{M}$,
— a function $\mathcal{F}_{\mathcal{M}} : \mathcal{S}_{\mathcal{M}} \to \mathcal{S}_{\mathcal{M}}$ called the *transition function* of $\mathcal{M}$,
— a function $\mathcal{R}_{\mathcal{M}} : \mathrm{PREM}_{\mathcal{M}} \to \mathrm{CONCL}_{\mathcal{M}}$ from a finite skeletal set $\mathrm{PREM}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ onto a finite skeletal set $\mathrm{CONCL}_{\mathcal{M}}$ of finite isomorphically perfect labelled directed graphs over $\Sigma_{\mathcal{M}}$ such that $\mathcal{R}_{\mathcal{M}}$ determines the *set*

$$\widetilde{\mathcal{R}}_{\mathcal{M}} = \{P \vdash C \mid P \in \mathrm{PREM}_{\mathcal{M}} \text{ and } C = \mathcal{R}_{\mathcal{M}}(P)\}$$

of rewriting rules of $\mathcal{M}$ which are identified with ordered pairs $r = (P_r, C_r)$, where the graph $P_r \in \mathrm{PREM}_{\mathcal{M}}$ is the *premise* of r and the graph $C_r = \mathcal{R}_{\mathcal{M}}(P_r)$ is the *conclusion* of r,
— a subset $\mathcal{I}_{\mathcal{M}}$ of $\mathcal{S}_{\mathcal{M}}$ which is the set of *initial instantaneous descriptions of* $\mathcal{M}$.

The above data are subject of the following conditions:

1) $V(\mathcal{G}) \subseteq V(\mathcal{F}_{\mathcal{M}}(\mathcal{G}))$ for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$,
2) $V(\mathcal{G}) \subseteq V(\mathcal{R}_{\mathcal{M}}(\mathcal{G}))$ for every $\mathcal{G} \in \mathrm{PREM}_{\mathcal{M}}$,
3) the rewriting rules of $\mathcal{M}$ are *applicable* to $\mathcal{S}_{\mathcal{M}}$ which means that for every $\mathcal{G} \in \mathcal{S}_{\mathcal{M}}$ the set

$$\begin{aligned}\mathcal{P}\ell(\mathcal{G}) = \big\{h \mid\ & h \text{ is an embedding of labelled graphs over } \Sigma \\ & \text{with } \mathrm{dom}(h) \in \mathrm{PREM}_{\mathcal{M}} \text{ and } \mathrm{cod}(h) = \mathcal{G} \\ & \text{such that for every embedding } h' \text{ of labelled graphs over } \Sigma \\ & \text{with } \mathrm{dom}(h') \in \mathrm{PREM}_{\mathcal{M}} \text{ and } \mathrm{cod}(h') = \mathcal{G} \\ & \text{if } \mathrm{im}(h) \text{ is a labelled subgraph of } \mathrm{im}(h'), \text{ then } h = h'\big\}\end{aligned}$$

of *maximal applications*[2] h of the rules $\mathrm{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h))$ of $\mathcal{M}$ in places $\mathrm{im}(h)$ is such that the following conditions hold:

[2] 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.

(i) $V(\mathcal{G}) = \bigcup_{h \in \mathcal{P}\ell(\mathcal{G})} V(\mathrm{im}(h))$, $E(\mathcal{G}) = \bigcup_{h \in \mathcal{P}\ell(\mathcal{G})} E(\mathrm{im}(h))$,

(ii) for all $h_1, h_2 \in \mathcal{P}\ell(\mathcal{G})$ the equation $\ell_{\mathcal{G}_{h_1}}(\dot{h}_1^{-1}(v)) = \ell_{\mathcal{G}_{h_2}}(\dot{h}_2^{-1}(v))$ holds for every $v \in V(\mathrm{im}(h_1)) \cap V(\mathrm{im}(h_2))$, where $\ell_{\mathcal{G}_{h_1}}$, $\ell_{\mathcal{G}_{h_2}}$ are the labelling functions of $\mathcal{G}_{h_1} = \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h_1))$, $\mathcal{G}_{h_2} = \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h_2))$, respectively, and $\dot{h}_1^{-1}$, $\dot{h}_2^{-1}$ are the inverses of isomorphisms induced by the embeddings h_1, h_2, respectively.

(iii) $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is a colimit of a gluing diagram $\mathcal{D}^{\mathcal{G}}$ constructed in the following way (the construction of $\mathcal{D}^{\mathcal{G}}$ is provided by (ii)):

- the set $\mathcal{I}$ of indexes of $\mathcal{D}^{\mathcal{G}}$ is such that $\mathcal{I} = \mathcal{P}\ell(\mathcal{G}) \cup \{\Delta\}$, where $\Delta \notin \mathcal{P}\ell(\mathcal{G})$ is the center of $\mathcal{D}^{\mathcal{G}}$,
- the family $\mathcal{G}_i$ $(i \in \mathcal{I})$ of labelled graphs of $\mathcal{D}^{\mathcal{G}}$ is such that $\mathcal{G}_h = \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h))$ for every $h \in \mathcal{P}\ell(\mathcal{G})$, and $\mathcal{G}_\Delta$ is such that $V(\mathcal{G}_\Delta) = V(\mathcal{G})$, $E(\mathcal{G}_\Delta) = \varnothing$, and the labelling function $\ell_{\mathcal{G}_\Delta}$ is such that provided by (ii)
$$\ell_{\mathcal{G}_\Delta}(v) = \ell_{\mathcal{G}_h}(\dot{h}^{-1}(v))$$
for every $v \in V(\mathrm{im}(h))$ and every $h \in \mathcal{P}\ell(\mathcal{G})$, where $\dot{h}^{-1}$ is the inverse of the isomorphism $\dot{h}$ induced by the embedding h,
- the gluing conditions gl_h $(h \in \mathcal{P}\ell(\mathcal{G}))$ of $\mathcal{D}^{\mathcal{G}}$ are defined by
$$\mathrm{gl}_h = \big\{(v, \dot{h}^{-1}(v)) \mid v \in V(\mathrm{im}(h))\big\}$$
for every $h \in \mathcal{P}\ell(\mathcal{G})$, where $\dot{h}^{-1}$ is the inverse of the isomorphism $\dot{h}$ induced by embedding h,

(iv) the following equations hold:
$$V(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) = \bigcup_{i \in \mathcal{I}} V(\mathrm{im}(q_i))$$
$$\text{and } E(\mathcal{F}_{\mathcal{M}}(\mathcal{G})) = \bigcup_{i \in \mathcal{I}} E(\mathrm{im}(q_i))$$
for the canonical injections $q_i : \mathcal{G}_i \to \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ $(i \in \mathcal{I})$ forming a colimiting cocone of the diagram $\mathcal{D}^{\mathcal{G}}$ defined in (iii),

(v) the canonical injection $q_\Delta : \mathcal{G}_\Delta \to \mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is an inclusion of labelled graphs, where Δ is the center of $\mathcal{D}^{\mathcal{G}}$ and q_Δ is an element of the colimiting cocone in (iv).

Thus $\mathcal{F}_{\mathcal{M}}(\mathcal{G})$ is the result of simultaneous application of the rules $\mathrm{dom}(h) \vdash \mathcal{R}_{\mathcal{M}}(\mathrm{dom}(h))$ in the places $\mathrm{im}(h)$ for $h \in \mathcal{P}\ell(\mathcal{G})$, where one replaces simultaneously $\mathrm{im}(h)$ by $\mathrm{im}(q_h)$ in $\mathcal{G}$ for $h \in \mathcal{P}\ell(\mathcal{G})$, respectively.

A finite sequence $\big(\mathcal{F}^i_{\mathcal{M}}(\mathcal{G})\big)_{i=0}^{n}$ is called a *finite computation of* $\mathcal{M}$, the number n is called the *time* of this computation, and $\mathcal{F}^n_{\mathcal{M}}(\mathcal{G})$ is called the *final instantaneous description* for this computation if

$$\mathcal{F}^0_{\mathcal{M}}(\mathcal{G}) = \mathcal{G} \in \mathcal{I}_{\mathcal{M}}, \quad \mathcal{F}^{n-1}_{\mathcal{M}}(\mathcal{G}) \neq \mathcal{F}^n_{\mathcal{M}}(\mathcal{G}), \quad \text{and } \mathcal{F}_{\mathcal{M}}(\mathcal{F}^n_{\mathcal{M}}(\mathcal{G})) = \mathcal{F}^n_{\mathcal{M}}(\mathcal{G}),$$

where $\mathcal{F}^i_{\mathcal{M}}(\mathcal{G})$ is defined inductively: $\mathcal{F}^i_{\mathcal{M}}(\mathcal{G}) = \mathcal{F}_{\mathcal{M}}\big(\mathcal{F}^{i-1}_{\mathcal{M}}(\mathcal{G})\big)$.

For a computation $\left(\mathcal{F}^i_{\mathcal{M}}(\mathcal{G})\right)_{i=0}^{n}$ its *space* is defined by

$$\mathrm{space}(\mathcal{M},\mathcal{G}) = \max\{\text{the number of elements of } V(\mathcal{F}^i_{\mathcal{M}}(\mathcal{G})) \mid 0 \le i \le n\}$$

for $\mathcal{G} \in \mathcal{I}_{\mathcal{M}}$, where intuitively $\mathrm{space}(\mathcal{M},\mathcal{G})$ is understood as the size of hardware measured by the number of indecomposable processors[3] used in the computations.

Example 1 (G–P–R machine simulating the computations of a Turing machine). Let $\mathbb{T}$ be a Turing machine[4] whose alphabet Σ (including blank symbol) is disjoint with the set Q of states of $\mathbb{T}$ and let $\delta : \Sigma \times Q \to \Sigma \times Q \times \{L, 0, R\}$ be the transition of $\mathbb{T}$ with cursor directions L for "left", 0 for "stay", and R for "right". We define a *graphical instantaneous description of* $\mathbb{T}$ to be a labelled directed graph $\mathcal{G}$ over $\Sigma^0 = \Sigma \cup Q \cup \{\%, \S\}$ with $\{\%, \S\} \cap (\Sigma \cup Q) = \varnothing$ such that

- $\mathcal{G}$ is induced by some acceptable ordered triple of integers, see Appendix,
- if $\mathcal{G}$ is induced by an acceptable ordered triple (k, m, n) of integers, then $\ell_{\mathcal{G}}(-k) = \%$, $\ell_{\mathcal{G}}(0) \in Q$, $\ell_{\mathcal{G}}(n) = \S$ and $\ell_{\mathcal{G}}(j) \in \Sigma$ for every $j \in \{i \in V(\mathcal{G}) \mid -k < i < n \text{ and } i \ne 0\}$ (here m corresponds to cursor position on Turing machine tape indicated by the edge $(0, m)$).

By Lemma 4 in Appendix the set $\mathcal{S}_{\mathbb{T}}$ of all graphical instantaneous descriptions of $\mathbb{T}$ is a skeletal set of isomorphically perfect labelled graphs. Thus we define a G–P–R machine $\mathcal{M}_{\mathbb{T}}$ aimed to simulate the computations of $\mathbb{T}$ such that

- the set of instantaneous descriptions of $\mathcal{M}_{\mathbb{T}}$ is the set $\mathcal{S}_{\mathbb{T}}$ of graphical instantaneous descriptions of $\mathbb{T}$,
- the transition function $\mathcal{F}_{\mathbb{T}}$ of $\mathcal{M}_{\mathbb{T}}$ and the rewriting rules of $\mathcal{M}_{\mathbb{T}}$ are determined by the transition function δ of $\mathbb{T}$ such that if $\delta(a, q) = (a', q', R)$, then

(f^R) if $\mathcal{G} \in \mathcal{S}_{\mathbb{T}}$ and $\mathcal{G}$ is induced by (k, m, n) such that $\ell_{\mathcal{G}}(m) = a$, $\ell_{\mathcal{G}}(0) = q$, then

1) if $m < n-1$, then $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ is that $\mathcal{G}'$ which is induced by $(k, \widehat{m}, n)$ with $\widehat{m} = m+1$ for $m \ne -1$ and $\widehat{m} = 1$ for $m = -1$ such that $\ell_{\mathcal{G}'}(0) = q'$, $\ell_{\mathcal{G}'}(m) = a'$, and $\ell_{\mathcal{G}'}(i) = \ell_{\mathcal{G}}(i)$ for every $i \in V(\mathcal{G}) - \{0, m\}$,
2) if $m = n-1$, then $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ is that $\mathcal{G}'$ which is induced by $(k, m+1, n+1)$ such that $\ell_{\mathcal{G}'}(0) = q'$, $\ell_{\mathcal{G}'}(m) = a'$, $\ell_{\mathcal{G}'}(n)$ is blank symbol, and $\ell_{\mathcal{G}'}(i) = \ell_{\mathcal{G}}(i)$ for every $i \in V(\mathcal{G}') - \{0, m, n\}$,

(r^R) the rewriting rules are given by the following two schemes $\mathcal{G}_p \vdash \mathcal{G}_c$ such that

(r^R_1) the premise $\mathcal{G}_p$ is such that $V(\mathcal{G}_p) = \{-1, 0, 1, 2\}$, $E(\mathcal{G}_p) = \mathrm{lin}[1, 2] \cup \{(0, 1)\}$, $\ell_{\mathcal{G}_p}(-1) \in \Sigma \cup \{\%\}$, $\ell_{\mathcal{G}_p}(0) = q$, $\ell_{\mathcal{G}_p}(1) = a$, $\ell_{\mathcal{G}_p}(2) \in \Sigma$, the conclusion $\mathcal{G}_c$ is such that $V(\mathcal{G}_c) = V(\mathcal{G}_p)$, $E(\mathcal{G}_c) = \mathrm{lin}[1, 2] \cup \{(0, 2)\}$, $\ell_{\mathcal{G}_c}(-1) = \ell_{\mathcal{G}_p}(-1)$, $\ell_{\mathcal{G}_c}(0) = q'$, $\ell_{\mathcal{G}_c}(1) = a'$, and $\ell_{\mathcal{G}_c}(2) = \ell_{\mathcal{G}_p}(2)$.

[3] The indecomposable processors coincide with urelements appearing in those Gandy machines which represent G–P–R machines in [5].

[4] For the definition and unexplained terms we refer the reader to, e.g., [6].

(r_2^R) the premise $\mathcal{G}_p$ is such that $V(\mathcal{G}_p)=\{-1,0,1,2\}$, $E(\mathcal{G}_p)=\mathrm{lin}[1,2]\cup\{(0,1)\}$, $\ell_{\mathcal{G}_p}(-1)\in\Sigma\cup\{\%\}$, $\ell_{\mathcal{G}_p}(0)=q$, $\ell_{\mathcal{G}_p}(1)=a$, $\ell_{\mathcal{G}_p}(2)=\S$, the conclusion $\mathcal{G}_c$ is such that $V(\mathcal{G}_c)=\{-1,0,1,2,3\}$, $E(\mathcal{G}_c)=\mathrm{lin}[1,3]\cup\{(0,2)\}$, $\ell_{\mathcal{G}_c}(-1)=\ell_{\mathcal{G}_p}(-1)$, $\ell_{\mathcal{G}_c}(0)=q'$, $\ell_{\mathcal{G}_c}(1)=a'$, $\ell_{\mathcal{G}_c}(2)$ is blank symbol, and $\ell_{\mathcal{G}_c}(3)=\S$.

For the cases of equations $\delta(a,q)=(a',q',0)$ and $\delta(a,q)=(a',q',L)$ the values $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ and the rewriting rules are defined in a similar way, where, e.g., the counterpart of (f_2^R) for $\delta(a,q)=(a',q',L)$ is:

(f_2^L) if $1=m=k$ or $-k+1=m\neq 0$, then $\mathcal{F}_{\mathbb{T}}(\mathcal{G})$ is that $\mathcal{G}'$ which is induced by $(k+1,-k,n)$ such that $\ell_{\mathcal{G}'}(-k-1)=\%$, $\ell_{\mathcal{G}'}(-k)$ is blank symbol, $\ell_{\mathcal{G}'}(0)=q'$, $\ell_{\mathcal{G}'}(m)=a'$, and $\ell_{\mathcal{G}'}(i)=\ell_{\mathcal{G}}(i)$ for all $i\in V(\mathcal{G}')-\{-k-1,-k,0,m\}$.

The versions of the above rules $\mathcal{G}_p\vdash\mathcal{G}_c$ for both $\mathcal{G}_p$ and $\mathcal{G}_c$ completed by the loop (i,i) for a unique $i\in V(\mathcal{G}_p)$ with $\ell_{\mathcal{G}_p}(i)\notin\{\%,\S\}\cup Q$ are also necessary. The identity rules $\mathcal{G}\vdash\mathcal{G}$ are also necessary, where $\mathcal{G}$ is of the following two forms:

(id_1) $V(\mathcal{G})=\{0,1\}$, $E(\mathcal{G})=\{(0,1)\}$, $\{\ell_{\mathcal{G}}(0),\ell_{\mathcal{G}}(1)\}\subset\Sigma^0-Q$,
(id_2) $V(\mathcal{G})=\{0\}$, $E(\mathcal{G})=\{(0,0)\}$, $\ell_{\mathcal{G}}(0)\in\Sigma$.

There is no other rewriting rule of $\mathcal{M}_{\mathbb{T}}$ than that described by the above schemes.

Since the graphical instantaneous descriptions of a Turing machine $\mathbb{T}$ coincide with the usual instantaneous descriptions of $\mathbb{T}$ or configurations of $\mathbb{T}$ as in [6], the G–P–R machine $\mathcal{M}_{\mathbb{T}}$ simulates the computations of $\mathbb{T}$ due to definition of $\mathcal{F}_{\mathbb{T}}$.

Example 2 (G–P–R machine simulating the computations of certain Boolean circuits). We define a *disjunctive circuit* G–P–R *machine* $\mathcal{M}_{\mathrm{circ}}$ which is aimed to simulate computations of certain tree like Boolean circuits such that

- the set $\mathcal{S}_{\mathrm{circ}}$ of instantaneous descriptions of $\mathcal{M}_{\mathrm{circ}}$ is the set of those regular labelled binary trees $\mathcal{T}$ of depth greater than 3 over the set $\{\mathrm{root},0,1\}\times\{\bot,0,1\}$ of labels, see Appendix, which satisfy the following condition
  (circ_0) for every binary string $\Gamma\in V(\mathcal{T})$ of length equal to the depth of $\mathcal{T}$ the number of elements of the set
  $$\big\{i \mid i \text{ is a natural number with } 0<i\leq n \text{ such that } \ell^2_{\mathcal{T}}(\Gamma\restriction i)\neq\bot\big\}$$
  is not greater than 1 (thus this set may be empty), where n is the depth of $\mathcal{T}$ and if Γ is $(k_j)_{j=1}^n$ then $\Gamma\restriction i$ denotes the string $(k_j)_{j=1}^i$ which is Γ itself for $i=n$ and for $i<n$ $(k_j)_{j=1}^i$ is a shortening of Γ by cancellation of the elements $k_n,k_{n-1},\ldots,k_{i+1}$.
- the transition function $\mathcal{F}_{\mathrm{circ}}$ of $\mathcal{M}_{\mathrm{circ}}$ is such that $\mathcal{F}_{\mathrm{circ}}(\mathcal{T})$ is the result of simultaneous application to $\mathcal{T}$ in G–P–R machine mode[5] the rewriting rules

[5] Understood that the result of simultaneous application is a colimit of the gluing diagram determined by the set $\mathcal{P}\ell(\mathcal{G})$ of maximal applications as in definition of G–P–R machine.

of $\mathcal{M}_{\mathrm{circ}}$ which do not introduce new vertices and which are given by the following three schemes $\mathcal{T}_p \vdash \mathcal{T}_c$ such that

(circ_1) the premise $\mathcal{T}_p$ is such that $V(\mathcal{T}_p) = \{\Lambda, 0, 00, 01\}$,
$E(\mathcal{T}_p) = \big\{(\Lambda, 0), (0, 00), (0, 01), (00, 00), (01, 01)\big\}$,
$\ell^2_{\mathcal{T}_p}(\Lambda) = \ell^2_{\mathcal{T}_p}(0) = \bot$, $\{\ell^1_{\mathcal{T}_p}(\Lambda), \ell^1_{\mathcal{T}_p}(0)\} \subseteq \{0, 1\}$,
$\ell^1_{\mathcal{T}_p}(00) = 0$, $\ell^1_{\mathcal{T}_p}(01) = 1$, $\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\} \subseteq \{0, 1\}$,
the conclusion $\mathcal{T}_c$ is such that $V(\mathcal{T}_c) = V(\mathcal{T}_p)$, $E(\mathcal{T}_c) = E(\mathcal{T}_p)$,
$\ell_{\mathcal{T}_c}(\Lambda) = \ell_{\mathcal{T}_p}(\Lambda)$, $\ell_{\mathcal{T}_c}(0) = \big(\ell^1_{\mathcal{T}_p}(0), \max\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\}\big)$,
$\ell_{\mathcal{T}_c}(00) = (0, \bot)$, $\ell_{\mathcal{T}_c}(01) = (1, \bot)$,

(circ_2) the premise $\mathcal{T}_p$ is such that $V(\mathcal{T}_p) = \{\Lambda, 0, 00, 01, 000, 001, 010, 011\}$,
$E(\mathcal{T}_p) = \big\{(\Gamma, \Gamma i) \mid \{\Gamma, \Gamma i\} \subseteq V(\mathcal{T}_p) \text{ and } i \in \{0, 1\}\big\}$,
$\ell^2_{\mathcal{T}_p}(\Gamma) = \bot$ for all $\Gamma \in V(\mathcal{T}_p) - \{00, 01\}$,
$\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\} \subseteq \{0, 1\}$, $\{\ell^1_{\mathcal{T}_p}(\Lambda), \ell^1_{\mathcal{T}_p}(0)\} \subseteq \{0, 1\}$,
$\ell^1_{\mathcal{T}_p}(\Gamma i) = i$ for all $\Gamma \in \{0, 00, 01\}$ and $i \in \{0, 1\}$,
the conclusion $\mathcal{T}_c$ is such that $V(\mathcal{T}_c) = V(\mathcal{T}_p)$, $E(\mathcal{T}_c) = E(\mathcal{T}_p)$,
$\ell_{\mathcal{T}_c}(\Gamma) = \ell_{\mathcal{T}_p}(\Gamma)$ for every $\Gamma \in V(\mathcal{T}_c) - \{0, 00, 01\}$,
$\ell_{\mathcal{T}_c}(0) = \big(\ell_{\mathcal{T}_p}(0), \max\{\ell^2_{\mathcal{T}_p}(00), \ell^2_{\mathcal{T}_p}(01)\}\big)$,
$\ell_{\mathcal{T}_c}(\Gamma) = \big(\ell^1_{\mathcal{T}_p}(\Gamma), \bot\big)$ for every $\Gamma \in \{00, 11\}$,

(circ_3) the premise $\mathcal{T}_p$ is such that $V(\mathcal{T}_p) = \{\Lambda, 0, 1, 00, 01, 10, 11\}$,
$E(\mathcal{T}_p) = \big\{(\Gamma, \Gamma i) \mid \{\Gamma, \Gamma i\} \subseteq V(\mathcal{T}_p) \text{ and } i \in \{0, 1\}\big\}$,
$\ell^1_{\mathcal{T}_p}(\Gamma i) = i$ for all $\Gamma \in \{\Lambda, 0, 1\}$ and $i \in \{0, 1\}$,
$\ell^2_{\mathcal{T}_p}(\Gamma) = \bot$ for every $\Gamma \in V(\mathcal{T}_p) - \{0, 1\}$,
$\ell^1_{\mathcal{T}_p}(\Lambda) = \text{root}$, $\{\ell^2_{\mathcal{T}_p}(0), \ell^2_{\mathcal{T}_p}(1)\} \subseteq \{0, 1\}$,
the conclusion $\mathcal{T}_c$ is such that $V(\mathcal{T}_c) = V(T_p)$, $E(T_c) = E(\mathcal{T}_p)$,
$\ell^2_{\mathcal{T}_c}(\Gamma) = \ell^2_{\mathcal{T}_p}(\Gamma)$ for every $\Gamma \in V(\mathcal{T}_c) - \{\Lambda, 0, 1\}$,
$\ell_{\mathcal{T}_c}(\Gamma) = (\ell^1_{\mathcal{T}_p}, \bot)$ for every $\Gamma \in \{0, 1\}$,
and $\ell_{\mathcal{T}_c}(\Lambda) = \big(\text{root}, \max\{\ell^2_{\mathcal{T}_p}(0), \ell^2_{\mathcal{T}_p}(1)\}\big)$.

The identity rules $\mathcal{T} \vdash \mathcal{T}$ are also necessary which are defined in a similar way as in Example 1.

There is no other rewriting rule of $\mathcal{M}_{\mathrm{circ}}$ than that described by the above schemes.

The following lemma characterizes the computations of G–P–R machine $\mathcal{M}_{\mathrm{circ}}$.

Lemma 1. *Let $\mathcal{T} \in \mathcal{S}_{\mathrm{circ}}$ be a regular labelled binary tree such that for every binary string Γ of length equal to the depth of $\mathcal{T}$ there exists a natural number i with $i > 0$ such that $\ell^2_{\mathcal{T}}(\Gamma \restriction i) \neq \bot$. Then for*

$$n = \max\big\{i \mid i \text{ is the length of some binary string } \Gamma \in V(\mathcal{T}) \text{ with } \ell^2_{\mathcal{T}}(\Gamma) \neq \bot\big\}$$

the value $\mathcal{F}^n_{\mathrm{circ}}(\mathcal{T})$ is that regular labelled tree $\mathcal{T}'$ which is such that $V(\mathcal{T}') = V(\mathcal{T})$, $E(\mathcal{T}') = E(\mathcal{T})$, $\ell^2_{\mathcal{T}'}(\Gamma) = \bot$ for all $\Gamma \in V(\mathcal{T}') - \{\Lambda\}$ and $\ell^2_{\mathcal{T}'}(\Lambda) = \max\big\{\ell^2_{\mathcal{T}}(\Gamma) \mid \Gamma \in V(\mathcal{T}')$ and $\ell^2_{\mathcal{T}}(\Gamma) \neq \bot\big\}$, where $\mathcal{F}^n_{\mathrm{circ}}(\mathcal{T})$ is defined inductively by $\mathcal{F}^1_{\mathrm{circ}}(\mathcal{T}) = \mathcal{F}_{\mathrm{circ}}(\mathcal{T})$ and $\mathcal{F}^n_{\mathrm{circ}}(\mathcal{T}) = \mathcal{F}_{\mathrm{circ}}(\mathcal{F}^{n-1}_{\mathrm{circ}}(\mathcal{T}))$.

Example 3 (assembly of binary trees). The set $\mathcal{S}_{\mathrm{tree}}$ of labelled directed graphs $\mathcal{T}_n^j$ for natural numbers j, n with $0 \leq j \leq 3 \cdot n - 3$ (for the definition of $\mathcal{T}_n^j$ see Appendix) is the set of instantaneous descriptions of a G–P–R machine $\mathcal{M}_{\mathrm{tree}}$ whose transition function $\mathcal{F}_{\mathrm{tree}}$ is defined by

$$\mathcal{F}_{\mathrm{tree}}(\mathcal{T}_n^j) = \mathcal{T}_n^{j+1} \quad (j \geq 0,\ n \geq 0).$$

The rewriting rules of $\mathcal{M}_{\mathrm{tree}}$ are given by the modified versions of the schemes $(\dot{r}_1)$, $(\dot{r}_{10})$, $(\dot{r}_{11})$, $(\dot{r}_{12})$ (see Appendix), and the identity rules, where, e.g., the claimed modification of $(\dot{r}_1)$ is obtained by restricting the scheme to vertices 1, 2. By Lemmas 6 and 7 in Appendix the machine $\mathcal{M}_{\mathrm{tree}}$ starting with $\mathcal{T}_n^0$ stops after $3 \cdot n - 3$ steps with $\mathcal{T}_n^{3 \cdot n - 3}$ as the final result which is isomorphic to a regular labelled binary tree of depth n $(n \geq 0)$. The machine $\mathcal{M}_{\mathrm{tree}}$ is inspired by the similar constructs in [8], where the membrane division rules correspond to the above rewriting rules of $\mathcal{M}_{\mathrm{tree}}$.

3 Randomized Gandy-Păun-Rozenberg Machines and NP Complete Problems

We propose an open, with some loss of precision, definition of randomized G–P–R machines to provide various their applications like applications in modelling systems solving NP complete problems shown in the present paper and the future applications in simulation of quantum computer computations by randomized systems.

Definition. By a *randomized* G–P–R *machine* we understand a G–P–R machine whose initial instantaneous descriptions contain certain configurations or structures chosen at random. These randomly chosen configurations or structures cause an uncertainty of the final result of machine computations which is measured by an error probability.

The randomly chosen configurations or structures will be described more precisely for particular applications, respectively.

We use the following basic concepts for randomization of G–P–R machines discussed in the present paper.

Definitions. By a *ground random ternary sequence* of length n we understand a finite sequence $\Theta = (\sigma_i^{\Theta})_{i=1}^n$ of digits 0, 1, and a symbol @ with all elements $\sigma_i^{\Theta} = \{0, 1, @\}$ chosen at random and with $k > 0$ occurrences of @ in Θ such that possibly $k = f(n)$ for some f satisfying $\lim\limits_{n \to \infty} \frac{f(n)}{n} = 0$.

A ground random ternary sequence $\Theta = (\sigma_i^{\Theta})_{i=1}^n$ with k occurrences of @ in Θ gives rise to 2^k binary strings $\Gamma = (\sigma_i^{\Gamma})_{i=1}^n$ of length n which are obtained from Θ by replacing the occurrences of @ in Θ by the occurrences of digits 0 or 1, i.e., $\sigma_i^{\Gamma} \in \{0, 1\}$ for $\sigma_i^{\Theta} = @$ and $\sigma_i^{\Gamma} = \sigma_i^{\Theta}$ for $\sigma_i^{\Theta} \in \{0, 1\}$. These 2^k binary strings Γ obtained from Θ, called *binary strings generated* by Θ, are random binary strings with randomness inherited from Θ.

A ground random ternary sequence Θ of length n with k occurrences of @ can be constructed from e.g. two random binary strings Γ and Γ' of length n and $n-k$, respectively, such that @ is the i-th element of Θ iff 1 is the i-th element of Γ and deleting all occurrences of @ in Θ yields Γ'.

For all unexplained terms of logic and computational complexity theory, including Turing machines and the formulation of SAT and 3-SAT problems, we refer the reader to [6].

Example 4 (a randomized G–P–R machine solving 3-SAT problem in a polynomial time). We use a Turing machine $\dot{\mathbb{T}}$ such that for every formula φ in a conjunctive normal form as in 3-SAT problem and every truth assignment T for variables of φ the machine decides in the time $\leq n^{k_0}$ whether φ is valid for T, where the ordered pair (φ, T) is an input for $\dot{\mathbb{T}}$ from which the machine begins the computation, k_0 is some constant natural number, and n is the number of variables occurring in φ. We claim for $\dot{\mathbb{T}}$ that:

(A) if n is the number of variables occurring in φ, then any truth assignment T for variables of φ is represented by that binary string Γ of length n in the machine tape which is such that if the value "True" is assigned to the i-th variable of φ, then 1 is the i-th element of Γ, otherwise the i-th element of Γ is 0,

(A′) Turing machine $\dot{\mathbb{T}}$ is constructed from some simpler three-string or three-tape Turing machine 3-$\mathbb{T}$ according to the general construction in the proof of Theorem 2.1, p. 30 of [6], where the first tape of 3-$\mathbb{T}$ is an output tape, the second tape is an input tape containing some presentation of a truth assignment, and the third tape is an input tape containing some presentation of a formula. The machine 3-$\mathbb{T}$ reads only its input tapes and does not move its head or cursor to the left or right on output tape.

(B) for the G–P–R machine $\mathcal{M}_{\dot{\mathbb{T}}}$ simulating the computations of $\dot{\mathbb{T}}$ if we have that

(b_1) $\mathcal{G}_{\varphi,\Gamma}$ is that initial instantaneous description of $\mathcal{M}_{\dot{\mathbb{T}}}$ which coincides with the initial instantaneous description or initial configuration for input (φ, T) with T represented by the binary string Γ in the machine tape as in (A),

(b_2) $\mathcal{G} = \mathcal{F}^q_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma})$ is the final instantaneous description of $\mathcal{M}_{\dot{\mathbb{T}}}$ for the case of the final or halting state "stop" reached by $\dot{\mathbb{T}}$ after q steps of computation starting with input (φ, T) with T related to Γ as in (A), where $\mathcal{F}_{\dot{\mathbb{T}}}$ is the transition function of $\mathcal{M}_{\dot{\mathbb{T}}}$ and $\mathcal{F}^q_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma})$ is inductively defined: $\mathcal{F}^1_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma}) = \mathcal{F}_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma})$ and $\mathcal{F}^q_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma}) = \mathcal{F}_{\dot{\mathbb{T}}}(\mathcal{F}^{q-1}_{\dot{\mathbb{T}}}(\mathcal{G}_{\varphi,\Gamma}))$,

then

(b'_1) $\mathcal{G}_{\varphi,\Gamma}$ is a labelled directed graph induced by an acceptable ordered triple $(1, 1, m^0_\varphi)$ providing natural numbers m^-_φ, m^+_φ with $1 < m^-_\varphi < m^+_\varphi < m^0_\varphi$ and $m^+_\varphi - m^-_\varphi = n$, such that $\ell_{\mathcal{G}_{\varphi,\Gamma}}(m^-_\varphi + j)$ is the j-th element of Γ for every j with $1 \leq j \leq n$, where the numbers m^-_φ, m^+_φ are determined by the construction of $\dot{\mathbb{T}}$ from 3-$\mathbb{T}$ such that the essential content of the second tape of 3-$\mathbb{T}$, i.e. Γ itself, is written in the n squares $(m^-_\varphi + 1)$-th, ..., m^+_φ-th, respectively, of the tape of $\dot{\mathbb{T}}$,

(b_2') $\mathcal{G}$ is a labelled graph induced by some acceptable triple $(1,1,m_\varphi^0)$ of integers such that $\ell_{\mathcal{G}}(0)$ is the final state "stop" and $\ell_{\mathcal{G}}(1)=1$ if φ is valid for the truth assignment represented by Γ, otherwise $\ell_{\mathcal{G}}(1)=0$.

The shape of formulas in a conjunctive normal form in 3-SAT problem (it suffices to consider formulas of $n>3$ variables which are conjunctions of $\leq 2^3\cdot\binom{n}{3}$ nonrepetitive clauses, each disjunction of three literals containing different variables) provides that the machine 3-$\mathbb{T}$ reaches the final state in the time not greater than $2^3\cdot n^5$ steps for a formula of n variables, hence by Theorem 2.1, p. 30 of [6] the claimed machine $\dot{\mathbb{T}}$ reaches the final state in the time not greater than $2^6\cdot n^{10}$ steps for a formula of n variables.

For a function f defined on and valued in the set of natural numbers with $\lim\limits_{n\to\infty}\frac{f(n)}{n}=0$ we outline a construction of a randomized G–P–R machine $\mathcal{M}^f_{\text{3-SAT}}$ aimed to solve 3-SAT problem in a polynomial time.

We introduce now those classes of labelled directed graphs over $\Sigma^\bullet$ which we then use to define instantaneous descriptions of $\mathcal{M}^f_{\text{3-SAT}}$ for $\Sigma^\bullet$ equal to a disjoint union $\Sigma\,\dot\cup\, Q\,\dot\cup\,\{q_0,q_1,q_2,q_3,q_4\}\,\dot\cup\,\{\S,@\}\,\dot\cup\,(\{0,1,\bot,\text{root}\}\times\{0,1,\bot\})$, where Σ and Q are the alphabet and the set of states of the Turing machine $\dot{\mathbb{T}}$, respectively, where $\square\in\Sigma$ is used to denote empty square of the tape of the machine $\dot{\mathbb{T}}$ and the states q_{start}, 'stop' of $\dot{\mathbb{T}}$ are the starting state and the halting state of $\dot{\mathbb{T}}$, respectively.

For natural numbers j,m,m',n with $0\leq j\leq 3\cdot f(n)-3$, $1<m<m'$, $n>3$, and for characteristic function δ_n of the subset $\{3\cdot q-2 \mid 0<q<f(n)\}\cup\{3\cdot f(n)-3\}$ of the set $\{0,1,\dots,3\cdot f(n)-3\}$ we define a class $\mathcal{K}^{\delta_n(j)\cdot m,m'}_{n,j}$ to be the class of labelled directed graphs $\mathcal{G}$ over $\Sigma^\bullet$ which are such that

- $V(\mathcal{G})=\big(T^j_{f(n)}\times\{(0,0)\}\big)\cup\bigcup\limits_{(i,f(n))\in T^j_{f(n)}}\big(V^{(i,f(n))}\times\{(i,f(n))\}\big)$ for

$$V^{(i,f(n))}=\begin{cases}\{0,1,\dots,m'\} & \text{if } \ell^1_{\mathcal{T}^j_{f(n)}}\big((i,f(n))\big)\in\{0,\bot\} \text{ or } \delta_n(j)=0,\\ \{1,\dots,m\} & \text{if } \ell^1_{\mathcal{T}^j_{f(n)}}\big((i,f(n))\big)=1 \text{ and } \delta_n(j)=1,\end{cases}$$

  for $T^j_{f(n)}$ and $\mathcal{T}^j_{f(n)}$ see Appendix,
- $E(\mathcal{G})=\big\{\big((v,(0,0)),(v',(0,0))\big)\mid(v,v')\in E(\mathcal{T}^j_{f(n)})\big\}\cup$
  $\bigcup_{(i,f(n))\in T^j_{f(n)}}\Big(\big\{\big(((i,f(n)),(0,0)),(1,(i,f(n)))\big)\big\}\cup$
  $\big\{\big((q,(i,f(n))),(q+1,(i,f(n)))\big)\mid q>0 \text{ and } \{q,q+1\}\subset V^{(i,f(n))}\big\}\Big)\cup X_j$,
  where X_j is determined by j in the following way:
  (I) if $\delta_n(j)=0$ then

$$X_j=\big\{\big((0,(i,f(n))),(1,(i,f(n)))\big)\mid(i,f(n))\in T^j_{f(n)}\big\},$$

(II) if $\delta_n(j) = 1$ and $j < 3 \cdot f(n) - 3$, then there exists a natural number p with $1 \le p < m$ such that

$$X_j = K_p^{\nabla} = \bigcup \Big\{ \nabla^n_{i,p,z} \mid \ell_{\mathcal{T}^j_{f(n)}}((i, f(n))) = 0,\ \ell_{\mathcal{T}^j_{f(n)}}((z, f(n))) = 1 \text{ and } \{(v, (i, f(n))), (v, (z, f(n)))\} \subset E(\mathcal{T}^j_{f(n)}) \text{ for some } v \Big\}$$

where $\nabla^n_{i,p,z}$ is such that

$$\nabla^n_{i,p,z} = \big\{ \big((p, (i, f(n))), (p, (z, f(n)))\big), \big((0, (i, f(n))), (p, (i, f(n)))\big), \big((0, (i, f(n))), (p, (z, f(n)))\big) \big\}.$$

(III) if $j = 3 \cdot f(n) - 3$, then one of the following conditions holds:

(III′) there exists a natural number p with $1 \le p < m$ such that $X_j = K_p^{\nabla}$,

(III″) $m = m'$ and there exists a function

$$g : \{(i, f(n)) \mid (i, f(n)) \in T^{3 \cdot f(n) - 3}_{f(n)}\} \to \{1, \ldots, m'\}$$

such that

$$X_j = \big\{ \big((0, (i, f(n))), (g((i, f(n))), (i, f(n)))\big) \mid (i, f(n)) \in T^{3 \cdot f(n) - 3}_{f(n)} \big\}.$$

For a formula φ in a conjunctive normal form of $n > 3$ variables and a ground random ternary sequence $\Theta = (\sigma_i^{\Theta})_{i=1}^n$ with $f(n)$ occurrences of @ in Θ we define an initial instantaneous description $\mathcal{G}^0_{\varphi,\Theta}$ of $\mathcal{M}^f_{\text{3-SAT}}$ to be a labelled directed graph belonging to $\mathcal{K}^{0,m^0_{\varphi}}_{n,0}$ such that

- $\big((0, (0, f(n))), (1, (0, f(n)))\big) \in E(\mathcal{G}^0_{\varphi,\Theta})$,
- $\ell_{\mathcal{G}^0_{\varphi,\Theta}}\big((v, (0, 0))\big) = \ell_{\mathcal{T}^0_{f(n)}}(v)$ for every $v \in T^0_{f(n)}$,
- $\ell_{\mathcal{G}^0_{\varphi,\Theta}}\big((0, (0, f(n)))\big) = q_0$,
- for r with $0 < r \le m^0_{\varphi}$

$$\ell_{\mathcal{G}^0_{\varphi,\Theta}}\big((r, (0, f(n)))\big) = \begin{cases} (\bot, \bot) & \text{if } r = 1, \\ \sigma_r^{\Theta} & \text{if } r = m^-_{\varphi} + q \text{ and } 1 \le q \le n, \\ \ell_{\mathcal{G}_{\varphi,\Gamma}}(r) & \text{otherwise,} \end{cases}$$

where $\mathcal{G}_{\varphi,\Gamma}$ is an initial instantaneous description of $\mathcal{M}_{\dot{\mathbb{T}}}$ for some Γ.

Thus the set $\mathcal{I}_{\mathcal{M}^f_{\text{3-SAT}}}$ of initial instantaneous descriptions of $\mathcal{M}^f_{\text{3-SAT}}$ is the set of labelled graphs of the form $\mathcal{G}^0_{\varphi,\Theta}$ for some φ, Θ as above, where the values $\ell_{\mathcal{G}^0_{\varphi,\Theta}}\big((r, (0, f(n)))\big)$ $(m^-_{\varphi} < r \le m^+_{\varphi})$ form a randomly chosen configuration in the initial instantaneous description $\mathcal{G}^0_{\varphi,\Theta}$ which makes $\mathcal{M}^f_{\text{3-SAT}}$ a randomized G–P–R machine.

For a formula φ and a ground random ternary sequence Θ as above we define *instantaneous descriptions of* $\mathcal{M}^f_{\text{3-SAT}}$ *in assembly phase* to be labelled directed graphs $\mathcal{G}^k_{\varphi,\Theta}$ belonging to $\mathcal{K}^{\delta_n(j)\cdot m, m^0_\varphi}_{n,j}$ and defined inductively such that $\mathcal{G}^k_{\varphi,\Theta}$ is the result of simultaneous application to $\mathcal{G}^{k-1}_{\varphi,\Theta}$ in G–P–R machine mode the rewriting rules given by the schemes $(\dot{r}_1)$–$(\dot{r}_{12})$ presented in Appendix, and the identity rules defined in a similar way as in Section 1.

For a formula φ and a ground random ternary sequence Θ as above we define *instantaneous descriptions of* $\mathcal{M}^f_{\text{3-SAT}}$ *in computation phase* to be labelled directed graphs $\dot{\mathcal{G}}^k_{\varphi,\Theta}$ $(k \geq 0)$ defined inductively such that

— $\dot{\mathcal{G}}^0_{\varphi,\Theta} \in \mathcal{I}^{\text{comp}}_{\varphi,\Theta} = \{\mathcal{G}^k_{\varphi,\Theta} \mid k > 0 \text{ and } \ell_{\mathcal{G}^k_{\varphi,\Theta}}((0,(i,f(n)))) = q_{\text{start}} \text{ for some } i\}$,

— $\dot{\mathcal{G}}^k_{\varphi,\Theta}$ is the result of simultaneous application to $\dot{\mathcal{G}}^{k-1}_{\varphi,\Theta}$ in G–P–R machine mode the rules of $\mathcal{M}_{\mathbb{T}}$ with % replaced by the elements of $\{0,1\}\times\{\perp\}$ and $\mathcal{M}_{\text{circ}}$, and the following new rules given by the scheme

$$\mathcal{G}_p \vdash \mathcal{G}_c, \qquad (*)$$

where the premise $\mathcal{G}_p$ is such that
$V(\mathcal{G}_p) = \{0,1,2,3,4\}$,
$E(\mathcal{G}_p) = \{(i,i+1) \mid \{i,i+1\} \subseteq V(\mathcal{G}_p) - \{0\}\} \cup \{(0,3),(2,2)\}$,
$\ell_{\mathcal{G}_p}(i) \in \{0,1,\perp\}\times\{\perp\}$ for every $i \in \{1,2\}$, $\ell_{\mathcal{G}_p}(3) \in \{0,1\}$, $\ell_{\mathcal{G}_p}(0) =$ "stop" $\in Q$, $\ell_{\mathcal{G}_p}(4) \in \Sigma$,
the conclusion $\mathcal{G}_c$ is such that $V(\mathcal{G}_c) = V(\mathcal{G}_p)$, $E(\mathcal{G}_c) = E(\mathcal{G}_p)$,
$\ell_{\mathcal{G}_c}(i) = \ell_{\mathcal{G}_p}(i)$ for every $i \in \{1,4\}$, $\ell_{\mathcal{G}_c}(0) = \ell_{\mathcal{G}_c}(3) = \perp$,
$\ell_{\mathcal{G}_c}(2) = (\ell^1_{\mathcal{G}_p}(2), \ell_{\mathcal{G}_p}(3))$, where $\ell^1_{\mathcal{G}_p}(2)$ is such that $\ell_{\mathcal{G}_p}(2) = (\ell^1_{\mathcal{G}_p}(2), \perp)$.

Thus the set $\mathcal{S}_{\mathcal{M}^f_{\text{3-SAT}}}$ of instantaneous descriptions of $\mathcal{M}^f_{\text{3-SAT}}$ is the union of the set of instantaneous descriptions of $\mathcal{M}^f_{\text{3-SAT}}$ in assembly phase and the set of instantaneous descriptions of $\mathcal{M}^f_{\text{3-SAT}}$ in computation phase. The transition function $\mathcal{F}^f_{\text{3-SAT}}$ of $\mathcal{M}^f_{\text{3-SAT}}$ is such that $\mathcal{F}^f_{\text{3-SAT}}(\mathcal{G}^m_{\varphi,\Theta}) = \mathcal{G}^{m+1}_{\varphi,\Theta}$ with $\mathcal{G}^m_{\varphi,\Theta} \notin \mathcal{I}^{\text{comp}}_{\varphi,\Theta}$, and $\mathcal{F}^f_{\text{3-SAT}}(\dot{\mathcal{G}}^m_{\varphi,\Theta}) = \dot{\mathcal{G}}^{m+1}_{\varphi,\Theta}$.

The set of rewriting rules of $\mathcal{M}^f_{\text{3-SAT}}$ does not contain any other rule than these introduced above for $\mathcal{M}^f_{\text{3-SAT}}$.

Theorem. *Let f be a computable function such that $\lim_{n\to\infty} \frac{f(n)}{n} = 0$ and let p_φ be an estimation of the probability that a formula φ as in* 3-SAT *problem is valid for a given truth assignment. Then the* G–P–R *machine $\mathcal{M}^f_{\text{3-SAT}}$ solves* 3-SAT *problem in a polynomial time with subexponential number of indecomposable processors determined by f and with error probability estimated by $(p_\varphi)^{(2^{f(n)})}$ where n is the number of variables occurring in φ.*

Proof. The proof is contained in (or can be extracted from) the following description of the computation of G–P–R machine $\mathcal{M}^f_{\text{3-SAT}}$.

The computation of $\mathcal{M}^{f}_{\text{3-SAT}}$ consists of two phases: the assembly phase preceding the computation phase such that one splits $\mathcal{M}^{f}_{\text{3-SAT}}$ into two G–P–R machines $\mathcal{M}^{f,1}_{\text{3-SAT}}$ and $\mathcal{M}^{f,2}_{\text{3-SAT}}$ corresponding to these phases, respectively, in the following way:

— the machine $\mathcal{M}^{f,1}_{\text{3-SAT}}$ corresponding to assembly phase and determined by the rules $(\dot{r}_1)$–$(\dot{r}_{12})$ begins its computation with the initial instantaneous description of $\mathcal{M}^{f}_{\text{3-SAT}}$ itself and assembles initial instantaneous descriptions of the machine $\mathcal{M}^{f,2}_{\text{3-SAT}}$ by assembly some trees (see the rules $(\dot{r}_1)$, $(\dot{r}_{10})$–$(\dot{r}_{12})$) and copying appropriate subgraphs (see the rules $(\dot{r}_2)$–$(\dot{r}_9)$),

— the machine $\mathcal{M}^{f,2}_{\text{3-SAT}}$ corresponding to computation phase and determined by the rules of $\mathcal{M}_{\dot{\mathbb{T}}}$ with % replaced by the elements of $\{0,1\}\times\{\perp\}$, $\mathcal{M}_{\text{circ}}$, and the new rules $(*)$ introduced in the definition of $\mathcal{M}^{f}_{\text{3-SAT}}$ continues the computation to reach the final result of the computation of $\mathcal{M}^{f}_{\text{3-SAT}}$ itself, where the computation of $\mathcal{M}^{f,2}_{\text{3-SAT}}$ consists of the following two subphases:
  - a phase of simultaneous simulation of computations of appropriate number of copies of the Turing machine $\dot{\mathbb{T}}$ which precedes the following phase
  - a phase of simulation of computation of some Boolean circuit.

More precisely, for a given formula φ (as in 3-SAT problem) of $n > 3$ variables and a ground random ternary sequence Θ of length n with $f(n)$ occurrences of @ in Θ the machine $\mathcal{M}^{f,1}_{\text{3-SAT}}$ starting with the initial instantaneous description $\mathcal{G}^{0}_{\varphi,\Theta}$ assembles instantaneous description $\dot{\mathcal{G}}^{0}_{\varphi,\Theta}$ which is an initial instantaneous description of $\mathcal{M}^{f,2}_{\text{3-SAT}}$. Here $\dot{\mathcal{G}}^{0}_{\varphi,\Theta}$ contains, coded in some way, those $2^{f(n)}$ random binary strings of length n which are generated by Θ and represent $2^{f(n)}$ randomly chosen truth assignments for φ, respectively. Then $\mathcal{M}^{f,2}_{\text{3-SAT}}$ simultaneously simulates the computations of $2^{f(n)}$ copies of $\dot{\mathbb{T}}$, where $2^{f(n)}$ truth assignments for φ represented by $2^{f(n)}$ randomly chosen binary strings generated by Θ are the inputs together with φ for these $2^{f(n)}$ copies of $\dot{\mathbb{T}}$, respectively.

Then Boolean circuit part of $\mathcal{M}^{f}_{\text{3-SAT}}$ simulates the computation of tree-like Boolean circuit $\mathcal{C}$ of $2^{f(n)}$ input gates (see Lemma 1), where the underlying graph of $\mathcal{C}$ is a tree of depth $f(n)$ and all non-input gates of $\mathcal{C}$ are OR gates. The $2^{f(n)}$ input gates of $\mathcal{C}$ receive those inputs which are the output results of the computations of the above $2^{f(n)}$ copies of $\dot{\mathbb{T}}$, respectively. Here each input gate g is associated with that copy $\dot{\mathbb{T}}_g$ of $\dot{\mathbb{T}}$ for which g is connected with that unique vertex i of the final graphical instantaneous description of $\dot{\mathbb{T}}_g$ for which $(0,i)$ is an edge of this final graphical instantaneous description and i is labelled by the output result of $\dot{\mathbb{T}}_g$ with 0 labelled by the final or halting state of $\dot{\mathbb{T}}_g$. The inputs of $\mathcal{C}$ are simultaneously processed by $\mathcal{C}$ to give the output result in the root of the underlying graph of $\mathcal{C}$. The output result contained in the root yields an answer (with the error probability estimated in the Theorem) to a question whether there exists a truth assignment for φ such that φ is valid for this assignment. Therefore $\mathcal{M}^{f}_{\text{3-SAT}}$ solves 3-SAT problem in a polynomial time. □

4 Concluding Remarks

For a formula φ of n variables as in 3-SAT problem and a ground random ternary sequence Θ of length n with $f(n)$ occurrences of @ in Θ the G–P–R machine $\mathcal{M}^f_{\text{3-SAT}}$ simultaneously simulates the computations of $2^{f(n)}$ copies of the Turing machine $\dot{\mathbb{T}}$ for $2^{f(n)}$ randomly chosen instances of input data (i.e., the $2^{f(n)}$ randomly chosen truth assignments[6] for φ), respectively, where these randomly chosen instances of input data are assembled in a polynomial time by the machine $\mathcal{M}^f_{\text{3-SAT}}$. Thus the machine $\mathcal{M}^f_{\text{3-SAT}}$ shows that a randomized parallelization of computations or a parallelized randomization of these computations is possible to solve 3-SAT problem in a polynomial time. The assembly of $2^{f(n)}$ of randomly chosen instances of input data by $\mathcal{M}^f_{\text{3-SAT}}$ coincides with some simultaneous random choice of these $2^{f(n)}$ instances of input data. Thus the parallelized randomization is here a spatial randomization despite to a temporal or sequential randomization realized by repeating sequentially, i.e. step by step, an experiment consisting of a random choice of a single instance of input data then processed by, e.g., a single Turing machine, where the repeating of the experiment decreases an error probability.

We point out that the initial instantaneous descriptions of $\mathcal{M}^f_{\text{3-SAT}}$ are of the size depending linearly on the size of input data of a formula and a truth assignment.

The machine $\mathcal{M}^f_{\text{3-SAT}}$ is a biologically inspired construct, according to the ideas contained in [8], which illustrates a capability of (self)assembling of a distributed system which then realizes a process of massively parallel computation. One can include this capability to a paradigm of biologically inspired computing.

References

1. Chi, H., Jones, E.L.: Generating parallel quasirandom sequences via randomization. J. Parallel Distrib. Comput. 67, 876–881 (2007)
2. Gandy, R.: Church's thesis and principles for mechanisms. In: Barwise, J., et al. (eds.) The Kleene Symposium, pp. 123–148. North-Holland, Amsterdam (1980)
3. Rozenberg, G.: Handbook of graph grammars and computing by graph transformation, vol. 1. World Scientific, River Edge (1997); Ehrig, H. et al., Applications, languages and tools, vol. 2. World Scientific, River Edge (1999); Ehrig, H., et al., Concurrency, parallelism, and distribution, vol. 3. World Scientific, River Edge (1999)
4. Koblitz, N.: Algebraic Aspects of Cryptography, Berlin (1998)
5. Obtułowicz, A.: Gandy–Păun–Rozenberg machines. Romanian J. of Information Science and Technology 13, 181–196 (2010)
6. Papadimitriou, G.: Computational Complexity. Addison–Wesley, Reading (1994)
7. Păun, G.: P systems with active membranes: Attacking NP complete problems. Journal of Automata, Languages and Combinatorics 6, 75–90 (2000)
8. Păun, G.: Membrane Computing. An Introduction, Berlin (2002)
9. Sieg, W., Byrnes, J.: An abstract model for parallel computations: Gandy's Thesis. The Monist 82(1), 150–164 (1999)

[6] In different way than e.g. in [1].

Appendix

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 $(\mathcal{G}, \mathrm{h} : V(\mathcal{G}) \to V(\mathcal{G}'), \mathcal{G}')$ such that h is a function from $V(\mathcal{G})$ into $V(\mathcal{G}')$ which satisfies the following conditions:

(H_1) $(v, v') \in E(\mathcal{G})$ implies $(\mathrm{h}(v), \mathrm{h}(v')) \in E(\mathcal{G}')$ for all $v, v' \in V(\mathcal{G})$,
(H_2) $\ell_{\mathcal{G}'}(\mathrm{h}(v)) = \ell_{\mathcal{G}}(v)$ for every $v \in V(\mathcal{G})$.

If a triple $h = (\mathcal{G}, \mathrm{h} : V(\mathcal{G}) \to V(\mathcal{G}'), \mathcal{G}')$ is a homomorphism of a labelled directed graph $\mathcal{G}$ over Σ into a labelled directed graph $\mathcal{G}'$ over Σ, we denote this triple by $h : \mathcal{G} \to \mathcal{G}'$, we write $\mathrm{dom}(h)$ and $\mathrm{cod}(h)$ for $\mathcal{G}$ and $\mathcal{G}'$, respectively, according to category theory convention, and we write $h(v)$ for the value $\mathrm{h}(v)$.

A homomorphism $h : \mathcal{G} \to \mathcal{G}'$ of labelled directed graphs over Σ is an *embedding of* $\mathcal{G}$ *into* $\mathcal{G}'$, denoted by $h : \mathcal{G} \rightarrowtail \mathcal{G}'$, if the following condition holds:

(E) $h(v) = h(v')$ implies $v = v'$ for all $v, v' \in V(\mathcal{G})$.

An embedding $h : \mathcal{G} \rightarrowtail \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ is an *inclusion of* $\mathcal{G}$ *into* $\mathcal{G}'$, denoted by $h : \mathcal{G} \hookrightarrow \mathcal{G}'$, if the following holds:

(I) $h(v) = v$ for every $v \in V(\mathcal{G})$.

We say that a labelled directed graph $\mathcal{G}$ over Σ is a *labelled subgraph* of a labelled directed graph $\mathcal{G}'$ over Σ if there exists an inclusion $h : \mathcal{G} \hookrightarrow \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ.

For an embedding $h : \mathcal{G} \rightarrowtail \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ we define the *image* of h, denoted by $\mathrm{im}(h)$, to be a labelled directed graph $\widehat{\mathcal{G}}$ over Σ such that $V(\widehat{\mathcal{G}}) = \big\{h(v) \mid v \in V(\mathcal{G})\big\}$, $E(\widehat{\mathcal{G}}) = \big\{(h(v), h(v')) \mid (v, v') \in E(\mathcal{G})\big\}$, and the labelling function $\ell_{\hat{\mathcal{G}}}$ of $\widehat{\mathcal{G}}$ is the restriction of the labelling function $\ell_{\mathcal{G}'}$ of $V(\mathcal{G}')$ to the set $V(\widehat{\mathcal{G}})$, i.e., $\ell_{\hat{\mathcal{G}}}(v) = \ell_{\mathcal{G}'}(v)$ for every $v \in V(\widehat{\mathcal{G}})$.

A homomorphism $h : \mathcal{G} \to \mathcal{G}'$ of labelled directed graphs over Σ is an *isomorphism* of $\mathcal{G}$ into $\mathcal{G}'$ if there exists a homomorphism $h^{-1} : \mathcal{G}' \to \mathcal{G}$ of labelled directed graphs over Σ, called the inverse of h, such that the following conditons hold:

(Iz_1) $h^{-1}(h(v)) = v$ for every $v \in V(\mathcal{G})$,
(Iz_2) $h(h^{-1}(v)) = v$ for every $v \in V(\mathcal{G}')$.

We say that a labelled directed graph $\mathcal{G}$ over Σ *is isomorphic* to a labelled directed graph $\mathcal{G}'$ over Σ if there exists an isomorphism $h : \mathcal{G} \to \mathcal{G}'$ of labelled graphs $\mathcal{G}, \mathcal{G}'$ over Σ.

For an embedding $h : \mathcal{G} \rightarrowtail \mathcal{G}'$ of labelled directed graphs $\mathcal{G}, \mathcal{G}'$ over Σ we define a homomorphism $\dot{h} : \mathcal{G} \to \mathrm{im}(h)$ by $\dot{h}(v) = h(v)$ for every $v \in V(\mathcal{G})$. This homomorphism $\dot{h}$ is an isomorphism of $\mathcal{G}$ into $\mathrm{im}(h)$, called an *isomorphism deduced by* h.

For a labelled directed graph $\mathcal{G}$ over Σ, the *identity homomorphism* (or simply, *identity of* $\mathcal{G}$), denoted by $\mathrm{id}_{\mathcal{G}}$, is the homomorphism $h : \mathcal{G} \to \mathcal{G}$ such that $h(v) = v$ for every $v \in V(\mathcal{G})$.

We say that a labelled directed graph $\mathcal{G}$ over Σ is an *isomorphically perfect* labelled directed graph over Σ if the identity homomorphism $\mathrm{id}_{\mathcal{G}}$ is a unique isomorphism of labelled directed graph $\mathcal{G}$ into $\mathcal{G}$.

Lemma 2. *Let $\mathcal{G}$ be an isomorphically perfect labelled directed graph over Σ and let $h : \mathcal{G} \to \mathcal{G}'$, $h' : \mathcal{G} \to \mathcal{G}'$ be two isomorphisms of labelled graphs $\mathcal{G}, \mathcal{G}'$ over Σ. Then $h = h'$.*

We say that a set or a class $\mathcal{A}$ of labelled directed graphs over Σ is *skeletal* if for all labelled directed graphs $\mathcal{G}, \mathcal{G}'$ in $\mathcal{A}$ if they are isomorphic, then $\mathcal{G} = \mathcal{G}'$.

A *gluing diagram* $\mathcal{D}$ of labelled directed graphs over Σ is defined by:

— its *set $\mathcal{I}$ of indexes* with a distinguished index $\Delta \in \mathcal{I}$, called the *center* of $\mathcal{D}$,
— its *family $\mathcal{G}_i$ $(i \in \mathcal{I})$ of labelled directed graphs* over Σ,
— its *family* gl_i $(i \in \mathcal{I} - \{\Delta\})$ *gluing conditions* which are sets of ordered pairs such that
  (i) $\mathrm{gl}_i \subseteq V(\mathcal{G}_\Delta) \times V(\mathcal{G}_i)$ for every $i \in \mathcal{I} - \{\Delta\}$,
  (ii) $(v, v') \in \mathrm{gl}_i$ implies $\ell_{\mathcal{G}_\Delta}(v) = \ell_{\mathcal{G}_i}(v')$ for all $v \in V(\mathcal{G}_\Delta)$, $v' \in V(\mathcal{G}_i)$, and for every $i \in \mathcal{I} - \{\Delta\}$,
  (iii) for every $i \in \mathcal{I} - \{\Delta\}$ if gl_i is non-empty, then there exists a bijection

$$b_i : L(\mathrm{gl}_i) \to R(\mathrm{gl}_i)$$

for $L(\mathrm{gl}_i) = \{v \mid (v, v') \in \mathrm{gl}_i \text{ for some } v'\}$ and $R(\mathrm{gl}_i) = \{v' \mid (v, v') \in \mathrm{gl}_i \text{ for some } v\}$ such that $\{(v, b_i(v)) \mid v \in L(\mathrm{gl}_i)\} = \mathrm{gl}_i$.

For a gluing diagram $\mathcal{D}$ of labelled directed graphs over Σ we define a *cocone* of $\mathcal{D}$ to be a family $h_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$ of homomorphisms of labelled directed graphs over Σ (here $\mathrm{cod}(h_i) = \mathcal{G}$ for every $i \in \mathcal{I}$) such that

$$l_{\mathcal{G}}(h_\Delta(v)) = l_{\mathcal{G}}(h_i(v'))$$

for every pair $(v, v') \in \mathrm{gl}_i$ and every $i \in \mathcal{I} - \{\Delta\}$.

A cocone $q_i : \mathcal{G}_i \to \widetilde{\mathcal{G}}$ $(i \in \mathcal{I})$ of $\mathcal{D}$ is called a *colimiting cocone of* $\mathcal{D}$ if for every cocone $h_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$ of $\mathcal{D}$ there exists a unique homomorphism $h : \widetilde{\mathcal{G}} \to \mathcal{G}$ of labelled directed graphs $\widetilde{\mathcal{G}}, \mathcal{G}$ over Σ such that $h(q_i(v)) = h_i(v)$ for every $v \in V(\mathcal{G}_i)$ and for every $i \in \mathcal{I}$. The labelled directed graph $\widetilde{\mathcal{G}}$ is called a *colimit* of $\mathcal{D}$, the homomorphisms q_i $(i \in \mathcal{I})$ are called *canonical injections* and the unique homomorphism h is called the *mediating morphism* for $h_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$.

For a gluing diagram $\mathcal{D}$ one constructs its colimit $\widetilde{\mathcal{G}}$ in the following way:

— $V(\widetilde{\mathcal{G}}) = \bigcup_{i\in\mathcal{I}} (V_i \times \{i\})$, where
$V_\Delta = V(\mathcal{G}_\Delta)$ for the center Δ of $\mathcal{D}$,
$V_i = V(\mathcal{G}_i) - R(\mathrm{gl}_i)$ for every $i \in \mathcal{I} - \{\Delta\}$,
— $E(\widetilde{\mathcal{G}}) = \bigcup_{i\in\mathcal{I}} E_i$, where
$E_\Delta = \big\{\big((v,\Delta),(v',\Delta)\big) \mid (v,v') \in E(\mathcal{G}_\Delta)\big\}$ for the center Δ of $\mathcal{D}$,
$E_i = \big\{\big((v,i),(v',i)\big) \mid (v,v') \in E(\mathcal{G}_i) \text{ and } \{v,v'\} \subseteq V_i\big\}$
$\cup \big\{\big((v,\Delta),(v',\Delta)\big) \mid (v,v'') \in \mathrm{gl}_i,\ (v',v''') \in \mathrm{gl}_i,$
and $(v'',v''') \in E(\mathcal{G}_i)$ for some $v'', v'''\big\}$
$\cup \big\{\big((v,\Delta),(v',i)\big) \mid v' \in V_i,\ (v,v'') \in \mathrm{gl}_i \text{ and } (v'',v') \in E(\mathcal{G}_i) \text{ for some } v''\big\}$
$\cup \big\{\big((v,i),(v',\Delta)\big) \mid v \in V_i,\ (v,v'') \in \mathrm{gl}_i \text{ and } (v,v'') \in E(\mathcal{G}_i) \text{ for some } v''\big\}$
for every $i \in \mathcal{I} - \{\Delta\}$,
— the labelling function $\ell_{\widetilde{\mathcal{G}}}$ is defined by $\ell_{\widetilde{\mathcal{G}}}((v,i)) = \ell_{\mathcal{G}_i}(v)$ for every $(v,i) \in V(\widetilde{\mathcal{G}})$.

The definition of a colimiting cocone of a gluing diagram $\mathcal{D}$ provides that any other colimit of $\mathcal{D}$ is isomorphic to the colimit of $\mathcal{D}$ constructed above. Hence one proves the following lemma.

Lemma 3. *Let $\mathcal{D}$ be a gluing diagram of labelled graphs over Σ. Then for every colimiting cocone $q_i : \mathcal{G}_i \to \mathcal{G}$ $(i \in \mathcal{I})$ of $\mathcal{D}$ if $i' \neq i''$, then*

$$\big(V(\mathrm{im}(q_{i'})) - V(\mathrm{im}(q_\Delta))\big) \cap \big(V(\mathrm{im}(q_{i''})) - V(\mathrm{im}(q_\Delta))\big) = \varnothing$$

for all $i', i'' \in \mathcal{I} - \{\Delta\}$, where Δ is the center of $\mathcal{D}$ and the elements of nonempty $V(\mathrm{im}(q_i)) - V(\mathrm{im}(q_\Delta))$ with $i \neq \Delta$ are 'new' elements and the elements of $V(\mathrm{im}(q_\Delta))$ are 'old' elements.

We say that an ordered triple (k,m,n) of integers k,m,n is *acceptable* if $k > 0$, $m \neq 0$, $n > 1$, and $-k < m < n$. We define

$$\mathrm{lin}[k,n] = \big\{(i,i+1) \mid i \text{ is an integer such that } -k \leq i < -1 \text{ or } 1 \leq i < n\big\} \cup \{(-1,1)\}$$

for k,n as above.

Then we say that a labelled directed graph $\mathcal{G}$ over Σ having more than one label is *induced by an acceptable ordered triple* (k,m,n) if $\mathcal{G}$ is such that

— $V(\mathcal{G}) = \big\{i \mid i \text{ is an integer such that } -k \leq i \leq n\big\}$,
— $E(\mathcal{G}) = \mathrm{lin}[k,n] \cup \{(0,m),(1,1)\}$,
— $\ell_{\mathcal{G}}(0) \notin \{\ell_{\mathcal{G}}(-k), \ell_{\mathcal{G}}(m)\}$.

For a natural number $n > 0$ a *regular labelled binary tree of depth n over* $\{\mathrm{root},0,1\} \times \Sigma$ is defined to be a labelled directed graph $\mathcal{T}$ over $\{\mathrm{root},0,1\} \times \Sigma$ such that

— $V(\mathcal{T})$ is the set of binary strings of length not greater than n including empty string Λ,

— $E(\mathcal{T}) = \big\{(\Gamma, \Gamma i) \mid \{\Gamma, \Gamma i\} \subseteq V(\mathcal{T}) \text{ and } i \in \{0,1\}\big\}$
$\quad \cup \big\{(\Gamma, \Gamma) \mid \Gamma \text{ is a binary string of length } n\big\}$,
— the labelling function $\ell_{\mathcal{T}} : V(\mathcal{T}) \to \{\text{root}, 0, 1, \bot\} \times \Sigma$ of $\mathcal{T}$ is such that $\ell^1_{\mathcal{T}}(\Lambda) = \text{root}$, $\ell^1_{\mathcal{T}}(\Gamma i) = i$ for every binary string Γ and every $i \in \{0,1\}$ such that $\Gamma i \in V(\mathcal{T})$,

where $\ell^1_{\mathcal{T}}(x)$, $\ell^2_{\mathcal{T}}(x)$ denote the coordinates such that $\ell_{\mathcal{T}}(x) = (\ell^1_{\mathcal{T}}(x), \ell^2_{\mathcal{T}}(x))$ and Γi denotes that binary string Θ whose last element is the digit i, and Γ is that binary string which is the result of deleting the last element in Θ.

Lemma 4. *The set of labelled directed graphs over Σ induced by acceptable ordered triples of integers is a skeletal set of isomorphically perfect graphs for Σ having more than one label.*

Lemma 5. *The set of all regular binary trees of arbitrary depth over $\{\text{root}, 0, 1\} \times \Sigma$ is a skeletal set of isomorphically perfect graphs.*

We adopt the following notation for a set X of ordered pairs of natural numbers:

$$i \odot X = \{(m \cdot 2^i, n + i) \mid (m,n) \in X\} \text{ for a natural number } i,$$
$$(0,1) \oplus X = \{(m, n+1) \mid (m,n) \in X\},$$
$$(1,1) \oplus X = \{(m + 2^n, n+1) \mid (m,n) \in X\},$$

we recall that subtraction $\dot{-}$ of natural numbers is given by $m \dot{-} n = 0$ if $n \geq m$ and $m - n$ if $n \leq m$.

We define inductively the finite sets T^m_n of ordered pairs of natural numbers for natural numbers m, n by the following equations:

(t_0) $T^0_n = \{(0,i) \mid i \text{ is a natural number with } 0 \leq i \leq n\}$,
(t_1) $T^m_0 = T^0_0$,
(t_2) $T^{n-i}_n = T^0_i \cup (i \odot T^{n-i}_{n-i})$ for $0 < i < n$,
(t_3) $T^{n+i}_n = T^0_0 \cup \big((0,1) \oplus T^{(n \dot{-} 3)+i}_{n-1}\big) \cup \big((1,1) \oplus T^{(n \dot{-} 3)+i}_{n-1}\big)$ for $n > 0$ and $i \geq 0$.

Lemma 6. *For all natural numbers $n \geq 2$ and $i > 0$ we have $T^{3 \cdot n - 3 + i}_n = T^{3 \cdot n - 3}_n$.*

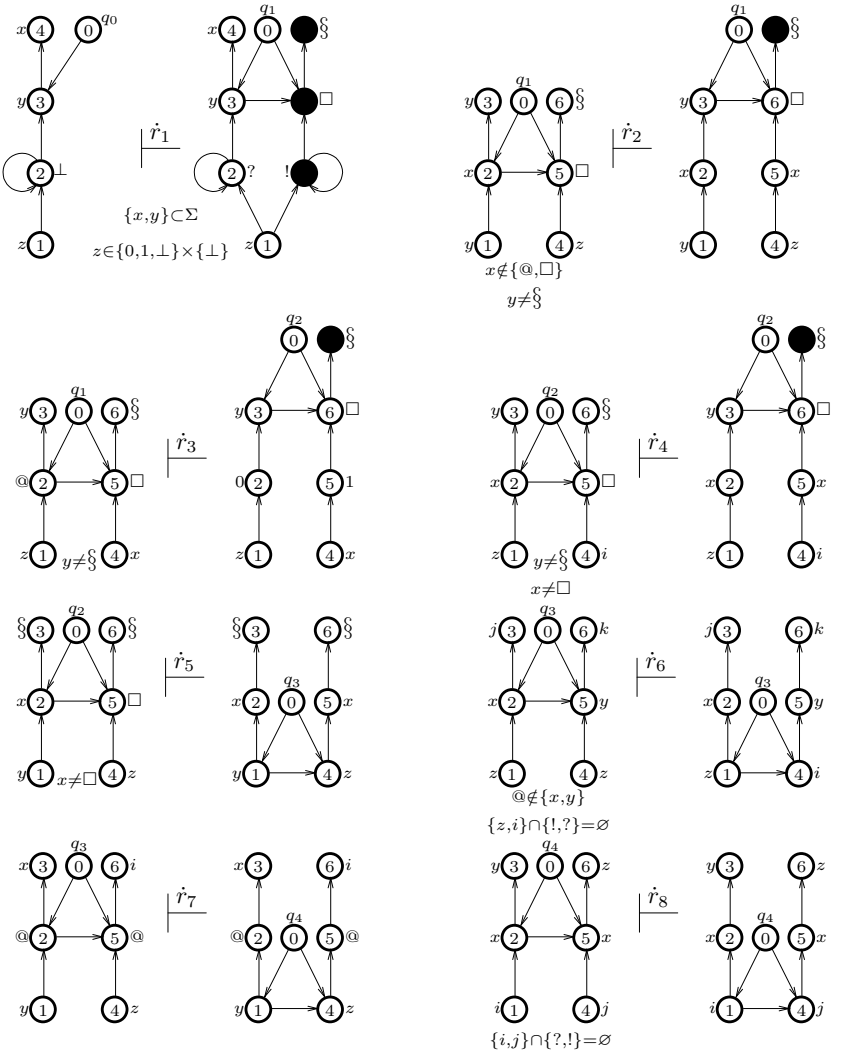
For natural numbers m, n we define labelled directed graph $\mathcal{T}^m_n$ over $\{\bot, \text{root}, 0, 1\} \times \{0, 1, \bot\}$ by

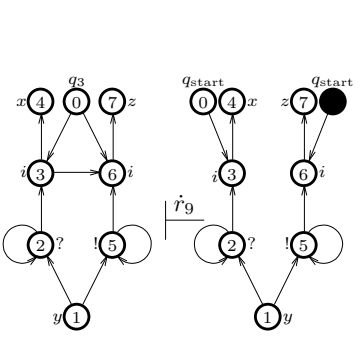
— $V(\mathcal{T}^m_n) = T^m_n$,
— $E(\mathcal{T}^m_n) = E^{m,n}_1 \cup E^{m,n}_2$ for
$E^{m,n}_1 = \big\{\big((i,j),(i',j')\big) \mid \{(i,j),(i',j')\} \subset T^m_n,\ j' = j+1,$
$i \leq i'/2$, and $i = \max\{k \mid (k,j) \in T^m_n,$ and $k \leq i'/2\}\big\}$,
$E^{m,n}_2 = \big\{\big((i,n),(i,n)\big) \mid (i,n) \in T^m_n\big\}$,
— the labelling function $\ell_{\mathcal{T}^m_n}$ of $\mathcal{T}^m_n$ is such that
$\ell_{\mathcal{T}^m_n}((i,j)) = \big(\ell^1_{\mathcal{T}^m_n}((i,j)), \bot\big)$ for $\ell^1_{\mathcal{T}^m_n}((i,j))$ defined by
(t'_0) $\ell^1_{\mathcal{T}^m_n}((0,0)) = \text{root}$,
(t'_1) if $x \in T^m_n$ with $x \neq (0,0)$ and $P_x = \big\{y \mid \{(z,y),(z,x)\} \subset E^{m,n}_1$ for some $z\big\}$ has exactly one element, then $\ell_{\mathcal{T}^m_n}(x) = \bot$,

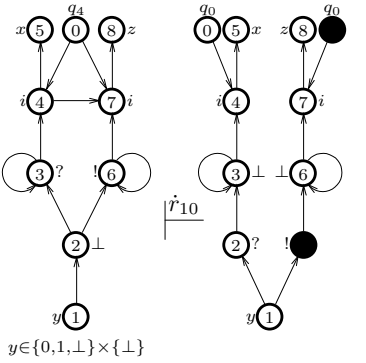
(t_2') if $x \in T_n^m$ and $P_x = \{(i,j),(i',j)\}$ with $i < i'$, then $\ell^1_{\mathcal{T}_n^m}((i,j)) = 0$ and $\ell^1_{\mathcal{T}_n^m}((i',j)) = 1$.

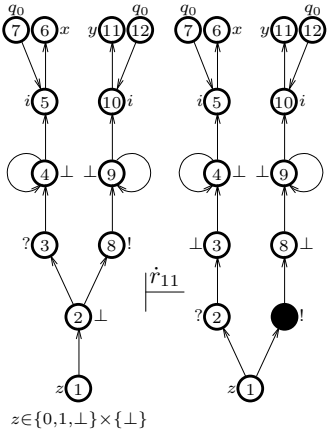
Lemma 7. *For every natural number $n \geq 2$ the labelled directed graph $\mathcal{T}_n^{3\cdot n-3}$ is isomorphic to a regular labelled binary tree $\mathcal{T}$ of depth n over $\{\bot, \mathrm{root}, 0, 1\} \times \{0, 1, \bot\}$ such that $\ell^2_{\mathcal{T}}(\Gamma) = \bot$ for all $\Gamma \in V(\mathcal{T})$.*

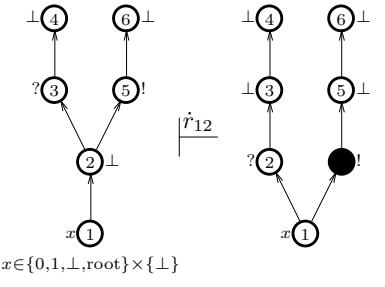
The schemes $(\dot{r}_1)$–$(\dot{r}_{12})$ of the rewriting rules are illustrated by the figures below, where the numbers in circles are the vertices of the corresponding labelled graphs, the new vertices are indicated by bold dots $\bullet$, the labels of the vertices stand close to the corresponding circles, and ?, !, $\bot$ are the abbreviations of the labels $(0,\bot)$, $(1,\bot)$, $(\bot,\bot)$, respectively. The variables x, y, z, i, j, k range over the set $\Sigma^\bullet$ of labels.




q_3
q_{start}
q_{start}
$\dot{r}_9$
$y\in\{0,1\}\times\{\bot\}$


q_4
q_0
q_0
$\dot{r}_{10}$
$y\in\{0,1,\bot\}\times\{\bot\}$


q_0
q_0
q_0
q_0
$\dot{r}_{11}$
$z\in\{0,1,\bot\}\times\{\bot\}$


$\dot{r}_{12}$
$x\in\{0,1,\bot,\mathrm{root}\}\times\{\bot\}$

Feasibility of Organizations – A Refinement of Chemical Organization Theory with Application to P Systems

Stephan Peter[1,*], Tomas Veloz[2,*], and Peter Dittrich[1]

[1] Friedrich Schiller University of Jena, Department for Mathematics and Computer Sciences, Bio Systems Analysis Group, D-07743 Jena, Germany
[2] University of Chile, Departamento de Ciencias de la Computacion, C.P. 837-0459 Santiago, Chile

Abstract. In membrane computing, a relatively simple set of reaction rules usually implies a complex "constructive" dynamics, in which novel molecular species appear and present species vanish. Chemical organization theory is a new approach that deals with such systems by describing chemical computing as a transition between organizations, which are closed and self-maintaining sets of molecular species. In this paper we show that for the case of mass-action kinetics some organizations are not feasible in the space of concentrations and thus need not to be considered in the analysis. We present a theorem providing criteria for an unfeasible organization. This is a refinement of organization theory making its statements more precise. In particular it follows for the design of a membrane computing system that the desired resulting organization of a chemical computing process should be a feasible organization. Nevertheless we show that due to the membranes in a P system unfeasible organizations can be observed, suggesting a strong link between the two approaches.

1 Introduction

Inspired by Fontana and Buss [6], Dittrich and Speroni have developed a general definition of a chemical organization [4]. Compared to other network analysis approaches like elementary flux modes [13,14], feedback loops [15], Petri nets [12] or deficiency [5], organization theory works on a very abstract level which allows for dealing with constructive dynamical systems. These systems allow for appearance and disappearance of qualitatively new variables beyond a quantitative change of a constant number of variables. They arise in particle physics, chemical systems or even in social systems [7] where communication can produce further, new communication. In those systems chemical organization theory can be applied for static [2,3] and dynamic [11] analyses, description [9] or design of chemical programs [8] despite a huge number of species effecting huge non-simulatable systems or incomplete information.

* Contributed equally.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 325–337, 2010.

Fixed points are important in the dynamical analysis of reaction systems [18]. Dittrich and Speroni have shown that every fixed point of a chemical reaction system corresponds to an organization (Theorem 1 in [4]). Given kinetic laws, there is not necessarily a fixed point for each organization. Thus, it is useful to rule out such unfeasible organizations. Assuming mass-action kinetics, in this paper we give a necessary and sufficient criterion for the feasibility of organizations. This criterion gives a refinement of the systems organizational structure allowing for a better description of its dynamical behavior.

Membrane computing or P systems [19,20] also deal with the understanding of chemical reaction systems, but its focus is different. They are concerned with reaction systems distributed to several compartments. We show that in a P system membranes can effect the occurrence of organizations, by making possible the stability of unfeasible organizations as well as disturbing the stability of feasible organizations. In Section 2 is provided the basic concepts of chemical organization theory, in Section 3 it is introduced the feasibility of organizations in chemical organization theory, the necessary and sufficient conditions to decide if an organization is feasible for mass-action kinetics systems and it is discussed the relevance of feasibility in P systems. In Section 4 there are presented some illustrative examples showing the consequences of feasibility in chemical organization theory and the tight relation between organizations, feasibility and membranes in the dynamics of a chemical system. Finally we conclude the work with the indication of future work and open questions.

2 Chemical Organization Theory

This chapter provides the basics of chemical organization theory. In what follows let $\mathcal{M} = \{s_1, \ldots, s_m\}$ be a finite set of m *species* reacting with each other according to a finite set $\mathcal{R} = \{r_1, \ldots, r_n\}$ of n *reactions*. Together, the set of species and the set of reactions is called *reaction network*.

2.1 Preliminaries

Given a vector $\mathbf{k} \in \mathbb{R}^n_{>0}$ containing a strictly positive rate constant for each reaction, we can describe the dynamics of the species concentrations $\mathbf{x}$ by the ODE

$$\dot{\mathbf{x}} = \mathbf{N}\mathbf{v}(\mathbf{x};\mathbf{k}), \tag{1}$$

where $\mathbf{N} \in \mathbb{Z}^{m\times n}$ is the stoichiometric matrix and $(\mathbf{v}(\mathbf{x};\mathbf{k}))_\mathbf{i} = \mathbf{k_i} \prod_{\mathbf{j=1}}^{\mathbf{m}} \mathbf{x_j^{a_{ij}}}$ for $i = 1, \ldots, n$, is the *flux* according to mass-action kinetics. Here x_j denominates the concentration of the species s_j. We call ODE (1) a *chemical reaction system*.

The number $a_{ij} \in \mathbb{N}$ denominates the number of occurences of s_j in the support (reactants) of the i-th reaction. Together these numbers form a matrix $\mathbf{A} \in \mathbb{N}^{n\times m}$. With the rate constants $\mathbf{k}$ on the diagonal of a diagonal matrix $\mathbf{K} \in \mathbb{N}^{n\times n}$ we can write

$$\mathbf{v}(\mathbf{x};\mathbf{k}) = \mathbf{K}\mathbf{x}^\mathbf{A}. \tag{2}$$

Definition 1. *For a reaction $r_i \in \mathcal{R}$, the set $supp(r_i) \equiv \{s_j \in \mathcal{M} : \ a_{ij} > 0\}$ is the **support** of r_i.*

Definition 2. *Let $\mathcal{P}(\mathcal{M})$ be the power set of $\mathcal{M}$ and*

$$\phi : \mathbb{R}^m_{\geq 0} \to \mathcal{P}(\mathcal{M}), \ \mathbf{x} \mapsto \phi(\mathbf{x}) \equiv \{s \in \mathcal{M} : \ x_s > 0\}. \tag{3}$$

*For a state $\mathbf{x} \in \mathbb{R}^m_{\geq 0}$, the set $\phi(\mathbf{x})$ is the **abstraction** of $\mathbf{x}$. For a given set of species $S \subseteq \mathcal{M}$, a state $\mathbf{x} \in \mathbb{R}^m_{\geq 0}$ is an **instance** of S if and only if its abstraction equals S.*

2.2 Chemical Organizations

The following definition is the core of chemical organization theory.

Definition 3. *A subset of species $S \subseteq \mathcal{M}$ is an **organization** if and only if*

1. *S is **closed**, i.e. none of the reactions with support within S produces a species which is not contained in S, and*
2. *S is **self-maintaining**, i.e. there is a flux vector $\mathbf{v} = (v_1, ..., v_n)$ with $\mathbf{Nv} \geq \mathbf{0}$ and*

$$v_i \begin{cases} > 0 \ \Leftrightarrow \ supp(r_i) \subseteq S, \\ = 0 \ otherwise. \end{cases}$$

The following Theorem from [4] relates fixed points to organizations.

Theorem 1. *If $\mathbf{x}$ is a fixed-point of the ODE (1), i.e. $\mathbf{Nv}(\mathbf{x}; \mathbf{k}) \geq \mathbf{0}$, then the abstraction $\phi(\mathbf{x})$ is an organization.*

Proofs can be found in [4] and [11].

Remark 1. Since fixed points play a crucial role in the analysis of dynamical systems [18], Theorem 1 provides a link between the behavior of a chemical reaction system and its underlying set of reactions. This justifies the study of the systems dynamics by chemical organization theory. In particular, organizations appear in the long-term behavior of chemical reaction systems making the observation of organizations in such systems provable [11].

Remark 2. The converse of Theorem 1 does not hold in general, i.e., given a chemical reaction system, the underlying reaction network can exhibit an organization O for which there is no fixed point with abstraction equal to O. This *unfeasibility* is studied in the next chapter.

3 Feasibility

In this section we introduce and study the feasibility of an organization. In what follows assume that $O \equiv \mathcal{M}$ is an organization.

3.1 Definitions

We are going to introduce the notations required to state the main theorem.

Definition 4. *O is **feasible with respect to** $\mathbf{k}$ if and only if there is a vector of concentrations $\mathbf{x} \in \mathbb{R}^m_{>0}$ such that*

$$\mathbf{N}\mathbf{v}(\mathbf{x};\mathbf{k}) \geq \mathbf{0}. \tag{4}$$

*O is **feasible** if and only if it is feasible with respect to each $\mathbf{k} \in \mathbb{R}^n_{>0}$. Otherwise it is **unfeasible**.*

Now we define the image $Im(\mathbf{A})$ of the matrix $\mathbf{A}$ and the set $Ker^*(\mathbf{N})$. The latter is derived from the kernel of the matrix $\mathbf{N}$ by only allowing for vectors with all components strictly positive and fulfilling an inequality instead of the equality required for the definition of the kernel.

Definition 5. *We define*

$$Im(\mathbf{A}) \equiv \{\mathbf{y} : \mathbf{y} = \mathbf{A}\mathbf{x},\, \mathbf{x} \in \mathbb{R}^m\}, \tag{5}$$
$$Ker^*(\mathbf{N}) \equiv \{\mathbf{v} \in \mathbb{R}^n_{>0} :\ \mathbf{N}\mathbf{v} \geq \mathbf{0}\}. \tag{6}$$

Next we define the application of the logarithm function to vectors and sets of vectors.

Definition 6. *For a set U of vectors, a vector $\mathbf{u} = (u_1, u_2, ...) \in U$ and a number $\beta > 0$ we define*

$$log_\beta(\mathbf{u}) \equiv (log_\beta(u_1), log_\beta(u_2), \ldots)^T, \tag{7}$$
$$log_\beta(U) \equiv \{\mathbf{w} :\ \mathbf{w} = log_\beta(\mathbf{u}),\, \mathbf{u} \in U\}, \tag{8}$$

where superscript T denotes vector transposition.

Lastly, we define arithmetic operations over sets of vectors.

Definition 7. *For sets U, V of vectors we define*

$$U + V \equiv \{\mathbf{w} :\ \mathbf{w} = \mathbf{u} + \mathbf{v},\, \mathbf{u} \in U, \mathbf{v} \in V\}. \tag{9}$$

3.2 Theorem

Now we state the main theorem which gives a necessary and sufficient criterion for feasibility in mass-action kinetics.

Theorem 2. *O is feasible if and only if*

$$\mathbb{R}^n \setminus (log_\beta(Ker^*(\mathbf{N})) - Im(\mathbf{A})) = \emptyset. \tag{10}$$

Proof. For any $\beta > 0$, O is feasible if and only if

$$\forall \mathbf{k} \in \mathbb{R}^n_{>0}\, \exists \mathbf{x} \in \mathbb{R}^m_{>0} : \mathbf{N}\mathbf{v}(\mathbf{x};\mathbf{k}) \geq \mathbf{0} \tag{11}$$

$$\overset{\text{Eq. 2}}{\iff}$$

$$\forall \mathbf{k} \in \mathbb{R}^n_{>0}\, \exists \mathbf{x} \in \mathbb{R}^m_{>0}, \hat{\mathbf{v}} \in \mathbb{R}^n_{>0} : \mathbf{N}\hat{\mathbf{v}} \geq \mathbf{0} \wedge \hat{\mathbf{v}} = \mathbf{v}(\mathbf{x};\mathbf{k}) = \mathbf{K}\mathbf{x}^{\mathbf{A}} \tag{12}$$

$$\overset{\text{Def. 5}}{\iff}$$

$$\forall \mathbf{k} \in \mathbb{R}^n_{>0}\, \exists \mathbf{x} \in \mathbb{R}^m_{>0}, \hat{\mathbf{v}} \in Ker^*(\mathbf{N}) : \hat{\mathbf{v}} = \mathbf{K}\mathbf{x}^{\mathbf{A}} \tag{13}$$

$$\iff$$

$$\forall \mathbf{k} \in \mathbb{R}^n_{>0}\, \exists \mathbf{x} \in \mathbb{R}^m_{>0}, \hat{\mathbf{v}} \in Ker^*(\mathbf{N}) : log_\beta(\hat{\mathbf{v}}) = log_\beta(\mathbf{k}) + \mathbf{A} \cdot log_\beta(\mathbf{x}) \tag{14}$$

$$\iff$$

$$\forall \mathbf{k} \in \mathbb{R}^n_{>0}\, \exists \mathbf{x} \in \mathbb{R}^m_{>0}, \hat{\mathbf{v}} \in Ker^*(\mathbf{N}) : log_\beta(\mathbf{k}) = log_\beta(\hat{\mathbf{v}}) - \mathbf{A} \cdot log_\beta(\mathbf{x}) \tag{15}$$

$$\overset{\hat{\mathbf{k}} \equiv log_\beta(\mathbf{k})}{\iff}$$

$$\forall \hat{\mathbf{k}} \in \mathbb{R}^n\, \exists \mathbf{x} \in \mathbb{R}^m_{>0}, \hat{\mathbf{v}} \in Ker^*(\mathbf{N}) : \hat{\mathbf{k}} = log_\beta(\hat{\mathbf{v}}) - \mathbf{A} \cdot log_\beta(\mathbf{x}) \tag{16}$$

$$\overset{\mathbf{y} \equiv \mathbf{A} \cdot log_\beta(\mathbf{x})}{\iff}$$

$$\forall \hat{\mathbf{k}} \in \mathbb{R}^n\, \exists \mathbf{y} \in Im(\mathbf{A}), \hat{\mathbf{v}} \in Ker^*(\mathbf{N}) : \hat{\mathbf{k}} = log_\beta(\hat{\mathbf{v}}) - \mathbf{y} \tag{17}$$

$$\overset{\mathbf{w} \equiv log_\beta(\hat{\mathbf{v}})}{\iff}$$

$$\forall \hat{\mathbf{k}} \in \mathbb{R}^n\, \exists \mathbf{y} \in Im(\mathbf{A}), \mathbf{w} \in log_\beta(Ker^*(\mathbf{N})) : \hat{\mathbf{k}} = \mathbf{w} - \mathbf{y}, \tag{18}$$

$$\iff$$

$$\mathbb{R}^n \setminus (log_\beta(Ker^*(\mathbf{N})) - Im(\mathbf{A})) = \emptyset. \tag{19}$$

Theorem 2 reveals how both the structure of the supports of the reactions (represented by $\mathbf{A}$) and the stoichiometric matrix $\mathbf{N}$ restrict the possible flux vectors. When the set of restrictions has a special structure the organization O is unfeasible. That means that there are rate constants such that there exists no set of strictly positive species concentrations to build up a flux vector $\mathbf{v}$ which is in $Ker^*(\mathbf{N})$.

Remark 3. The set $log_\beta(Ker^*(\mathbf{N}))$ is not a linear vector space with respect to addition. This makes the computational verification of Theorem 2 by methods from linear algebra difficult.

As a consequence of Theorem 2 we get the following statement about the existence of fixed points.

Corollary 1. *If the set* $\mathbb{R}^n \setminus (log_\beta(Ker^*(\mathbf{N})) - Im(\mathbf{A}))$ *is not empty, then there are rate constants* $\mathbf{k}$ *such that the ODE (1) has no fixed point with abstraction equal* O*.*

Proof. We assume that the set $\mathbb{R}^n \setminus (log_\beta(Ker^*(\mathbf{N})) - Im(\mathbf{A}))$ is not empty. Then from Theorem 2 follows that O is unfeasible. Then there exists a vector of rate constants $\mathbf{k}$ such that O is not feasible with respect to this $\mathbf{k}$ (Definition 4).

Also from Definition 4 follows that then for all vectors of concentrations $\mathbf{x} \in \mathbb{R}^m_{>0}$ the inequality

$$\mathbf{Nv}(\mathbf{x};\mathbf{k}) \geq \mathbf{0} \tag{20}$$

does not hold. Particularly, $\mathbf{Nv}(\mathbf{x};\mathbf{k}) \neq \mathbf{0}$ for all $\mathbf{x} \in \mathbb{R}^m_{>0}$, *i.e.*, there exists no fixed point with abstraction equal O for the ODE (1).

Remark 4. If the set $\mathbb{R}^n \setminus (log_\beta(Ker^*(\mathbf{N})) - Im(\mathbf{A}))$ is empty, there can also exist rate constants such that the ODE (1) has no fixed point with abstraction equal O. The reason is that $\mathbf{Nv}(\mathbf{x};\mathbf{k}) \geq \mathbf{0}$ does not imply $\mathbf{Nv}(\mathbf{x};\mathbf{k}) = \mathbf{0}$. And fixed points require strict equality.

3.3 Feasibility in P Systems

Now we know that the set of species representing an unfeasible organization O is less probable to be observed in a single reaction system than one representing a feasible organization. We will show in the next section that for a P system membranes can compensate for this phenomenon (cf. Example 5): a membrane can provide several compartments each possibly containing different subsets of species (feasible organizations) such that there union is O. In this case, the unfeasibility of organization O can be attributed to an *incompatibility* of two smaller feasible organizations. I.e., whenever an unfeasible organization O can be written as union of feasible organizations, membranes can allow for the appearance of O even though it is unfeasible, because the membranes helps to separate properly the incompatible organizations. Furthermore, we will show that, even when an exchange of molecules between the compartments is permitted, the unfeasible organization can be maintained in time. Thus, in a P system, membranes can allow for the appearance of those sets of species which - following the refinement of chemical organization theory stated in the previous sections - could not appear if there would not be a membrane. This implies that destruction of membranes can lead also to the opposite effect, destabilizing the equilibria process described above between unfeasible organizations (partially or totally) separed by the membranes.

4 Examples

In this section there are shown different aspects of unfeasibility as well as its relation with P systems. Therefore the notation of Theorem 2 is used. The following two examples show straight consequences of Theorem 2:

Example 1. Let $\mathcal{M} = \{s_1, s_2\}$ and $\mathcal{R} = \{s_1 \to s_2, s_2 \to s_1\}$. We have

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{N} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Note that

$$Im(\mathbf{A}) = \mathbb{R}^2, \quad Ker^*(\mathbf{N}) = \{(v_1, v_2) \in \mathbb{R}^2_{>0} : \ v_1 = v_2\}.$$

Then we have

$$\mathbb{R}^2 \setminus (log_\beta(Ker^*(\mathbf{N})) - Im(\mathbf{A})) = \emptyset.$$

Thus, $\mathcal{M}$ is feasible. We are going to verify this.
We have to solve the system

$$\mathbf{Nv}(\mathbf{x}) = \mathbf{NKx}^{\mathbf{A}} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} k_1 x_1 \\ k_2 x_2 \end{pmatrix} \geq 0. \tag{21}$$

Where k_i and x_i are the rate constant of reaction r_i and the concentration of species s_i for $i = 1, 2$, respectively. Then, $\mathbf{Nv}(\mathbf{x}) \geq 0$ iff $x_1 = \frac{k_2}{k_1} x_2$. The latter equation has a solution for all strictly positive reaction rates $\mathbf{k} = (k_1, k_2)$, thus we conclude $\mathcal{M}$ is feasible.

Example 2. Let $\mathcal{M} = \{s_1, s_2\}$ and $\mathcal{R} = \{s_1 + s_2 \to 2s_2, s_1 + s_2 \to 2s_1\}$. Then we have

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{N} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Note that

$$Im(\mathbf{A}) = \{(v_1, v_2) \in \mathbb{R}^2 : \ v_1 = v_2\} = Ker^*(\mathbf{N}) \tag{22}$$

$$\Rightarrow$$

$$Im(\mathbf{A}) - log_\beta(Ker^*(\mathbf{N})) = \{(v_1, v_2) \in \mathbb{R}^2 : \ v_1 = v_2\} \tag{23}$$

$$\Rightarrow$$

$$\mathbb{R}^n \setminus (Im(\mathbf{A}) - log_\beta(Ker^*(\mathbf{N}))) \neq \emptyset. \tag{24}$$

Thus, for this example $\mathcal{M}$ is unfeasible. We are going to verify this by analysing the inequalities

$$\mathbf{Nv}(\mathbf{x}) = \mathbf{NKx}^{\mathbf{A}} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} k_1 x_1 x_2 \\ k_2 x_1 x_2 \end{pmatrix} \geq 0. \tag{25}$$

They are solvable if and only if $k_1 x_1 x_2 = k_2 x_1 x_2$, i.e. $k_1 = k_2$. Thus, $\mathcal{M}$ is unfeasible, because $\mathcal{M}$ is not feasible with respect to any rate constant vector fulfilling $k_1 \neq k_2$.

Remark 5. Note that in Examples 1 and 2 it is obtained the same stoichiometric matrix from the reactions which define the systems. But in Example 1 the organization $\{s_1, s_2\}$ is feasible and in Example 2 the organization $\{s_1, s_2\}$ is not feasible. Then we conclude that feasibility is a phenomenon which is beyond the stoichiometric information.

Example 3. In this example it is shown a way to build unfeasible organizations. We will use a reaction network of four molecular species and four reactions, but the method we are going to exemplify could be done for any set of molecules and reactions. Let $\mathcal{M} = \{s_1, s_2, s_3, s_4\}$ and x_i the concentration of species s_i for $i = 1, ..., 4$. We are going to build $\mathcal{R}$ such that $\mathcal{M}$ is an unfeasible organization.

First we are going to choose $Ker^*(\mathbf{N})$ such that $log_\beta(Ker^*(\mathbf{N})) \subsetneqq \mathbb{R}^n$ (In other case $\mathcal{M}$ would be feasible without necessity of knowing $\mathbf{A}$). For simplicity we choose

$$Ker^*(\mathbf{N}) = \{(v, v, v, v) : \ v > 0\}.$$

Now we are going to choose the support of the reactions:

$$\begin{aligned} \mathbf{v}(\mathbf{x}) = (&k_1 x_1 x_2, \\ &k_2 x_1 x_3, \\ &k_3 x_3 x_4, \\ &k_4 x_4 x_2). \end{aligned} \tag{26}$$

Note there are two shared species between every triad of reactions, what helps to obtain unfeasibility. As we already know $Ker^*(\mathbf{N})$, we can choose some relation between the reaction rates in order to obtain contradictory concentration equations for every vector in $Ker^*(\mathbf{N})$ fulfilling mass-action kinetics (Equation 4). Note that to every flux vector which verifies the self-maintenance property of $\mathcal{M}$ has to hold that $k_1 x_1 x_2 = k_2 x_1 x_3 = k_3 x_3 x_4 = k_4 x_4 x_2$. We are going to prove that if $k_1 > k_2$ and $k_3 > k_4$ the organization is unfeasible with respect to $\mathbf{k}$. By the first and second reactions we have $k_1 x_1 x_2 = k_2 x_1 x_3$. As $k_1 > k_2$ we have

$$x_3 > x_2. \tag{27}$$

By third and fourth reaction we have $k_3 x_3 x_4 = k_4 x_4 x_2$. As $k_3 > k_4$ we have

$$x_2 > x_3. \tag{28}$$

We have a contradiction. Note that if we would choose $k_1 < k_2$ and $k_3 < k_4$ we would obtain also a contradiction. Now we just have to build up the stoichiometric matrix by choosing the produced species of each reaction, in order to keep $Ker^*(\mathbf{N})$ as it was stated at the beginning of the example. Choosing the reactions

$$\begin{aligned} s_1 + s_2 &\rightarrow 2s_1, \\ s_1 + s_3 &\rightarrow 2s_2, \\ s_3 + s_4 &\rightarrow 2s_3, \\ s_2 + s_4 &\rightarrow 2s_4, \end{aligned} \tag{29}$$

we obtain $Ker^*(\mathbf{N})$ as it was stated at the beginning of this example. Then the organization $\{s_1, s_2, s_3, s_4\}$ is unfeasible. Theorem 2 confirms this.

Remark 6. It is interesting that in Example 2 both reactions have the same support. Example 3 shows that even if all reactions have different supports, the organization can be unfeasible.

Remark 7. In Example 3, note that if we would have chosen $Ker^*(\mathbf{N}) = \{(v_1, v_2, v_3, v_4) : \ v_1 \geq v_2 \geq v_3 \geq v_4\}$ and the same support for the reactions, it would

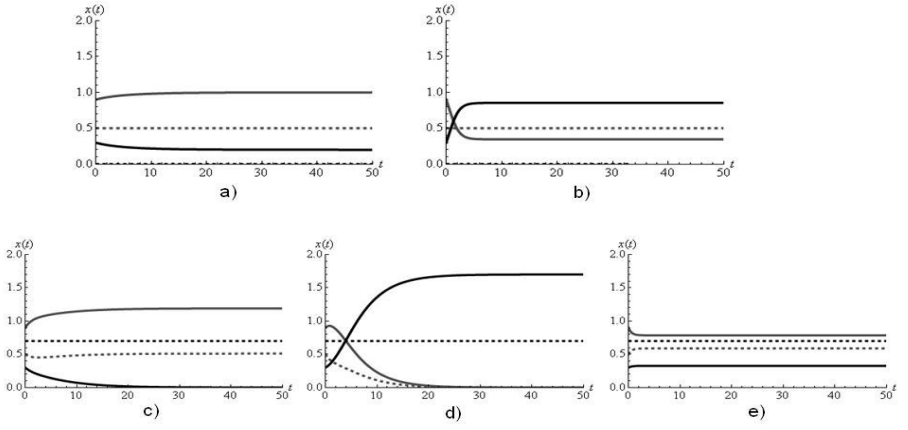


Fig. 1. Evolution of the concentrations in mass-action kinetics. The concentration of s_1, s_2, s_3, s_4 are grey, grey-dashed, black and black-dashed respectively. Figures **a**) and **b**) show the evolution in time beginning from an instance of the organization O_8 for two different sets of rate constants. Thus, despite changing the rate constants, the evolution of O_8 in **a**) and **b**) is structurally the same, asymptotic convergence to a fixed point corresponding to O_8, i.e. all species are *persistent*. This is possible since O_8 is feasible. Beginning from O_7 we obtain the same behavior (not shown in the figures) because of its feasibility. Figures **c**),**d**),**e**) show the evolution of concentrations beginning from an instance of $\mathcal{M}$ for three different sets of rate constants, such that $\mathcal{M}$ is unfeasible with respect to the rate constants chosen for Figures **c**) and **d**) and feasible with respect to the rate constants chosen for **e**). The system asymptotically tends to the reactive organization O_7 (Figure **c**)), the non-reactive organization $O_5 \equiv \{s_3, s_4\}$ (Figure **d**)) or $\mathcal{M}$ (Figure **e**)). Non-persistence of $\mathcal{M}$ in cases **c**) and **d**) is predicted by the unfeasibility of $\mathcal{M}$. The situation is illustrated within the lattice of organizations in Figure 2 (Left).

be obtained the same contradictions stated in Equations 27 and 28 when $k_1 > k_2$ and $k_3 > k_4$. In the opposite case, if we would have choosen $Ker^*(\mathbf{N}) = \{(v_1, v_2, v_3, v_4) : \ v_1 \leq v_2 \leq v_3 \leq v_4\}$ it would be obtained the contradictions when $k_1 < k_2$ and $k_3 < k_4$. In both cases mentioned above, the stoichiometric matrix which keeps $Ker^*(\mathbf{N})$ would look quite different to the stoichiometric matrix of Example 3.

Remark 8. The contradiction which we found between Equations 27 and 28 can be thought as a *game* where given $Ker^*(\mathbf{N})$, it has to be chosen the support of the reactions and the relation between reaction rates to obtain a system of inequalities $\mathbf{Nv} \geq \mathbf{0}$ with no solution.

The unfeasible organizations shown in the previous examples were composed by non-reactive organizations, this means any organization which is (strictly) contained in the unfeasible organization mentioned in every previous example verifies its self-maintenance by an empty flux vector.

Definition 8. *An organization is non-reactive if its self-maintenance is verified by an empty flux vector.*

From the point of view of combining organizations, what is shown in the previous examples is that combining non-reactive (trivially feasible) organizations it is possible to obtain an unfeasible organization. An interesting question is if it is possible to obtain an unfeasible organization from combining feasible and reactive organizations. The next example shows the positive answer to that question.

Example 4. In this example we are going to show that unfeasibility can be also generated by incompatibility between *reactive* organizations.

Let $\mathcal{M} = O_8 \cup O_7, O_8 = \{s_1, s_2, s_3\}, O_7 = \{s_1, s_2, s_4\}$, and $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$, where

$$r_1 : s_1 + s_3 \rightarrow 2s_3,$$
$$r_2 : s_2 + s_3 \rightarrow s_1 + s_2,$$
$$r_3 : s_2 + s_4 \rightarrow 2s_4,$$
$$r_4 : s_1 + s_4 \rightarrow s_1 + s_2.$$

Note that O_8 and O_7 are both feasible (and reactive) organizations, but together they form $\mathcal{M}$ which is an unfeasible organization. This is because under $k_1 < k_2$ and $k_3 > k_4$ (or $k_1 > k_2$ and $k_3 < k_4$) the flux vector which verifies self-maintenance of $\mathcal{M}$ has to hold $x_1 > x_2$ and $x_2 > x_1$ simultaneously[1].

For the results of simulations of this example see Figure 1 and Figure 2 (Left).

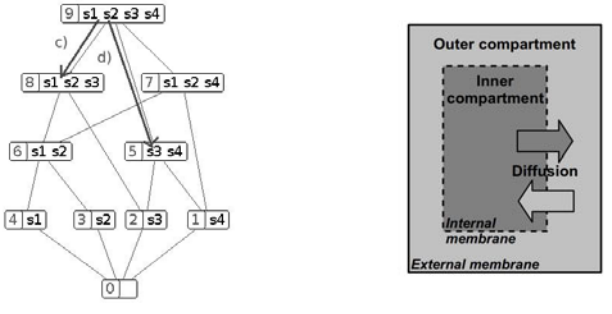


Fig. 2. Left: Lattice of organizations of Example 4. Each rectangle stands for an organization. There is a thin line between two organizations of different sizes iff the smaller organization is a subset of the bigger one and no other organization is between them. The arrows denote the (down-)movement in time from the unfeasible organization O_9 to the feasible organization O_8 (cf. Fig. 1 (c)), and to the non-reactive organization O_5 (cf. Fig. 1 (d)). **Right:** Diagram of the P system of Example 5. There are two compartments each surrounded by a membrane. The membranes allow for exchange-diffusion of molecules between compartments. Within each compartment the reaction rules are the same. But due to different initial conditions, different reaction rate for the same reaction in each compartment, or diffusion, the species concentrations can differ and different organizations can appear in each compartment.

[1] To verify this it is required to build $Ker^*(\mathbf{N})$ and check that under those reaction rates no flux vector can follow mass-action kinetics and simultaneously verify the self-maintenance property.

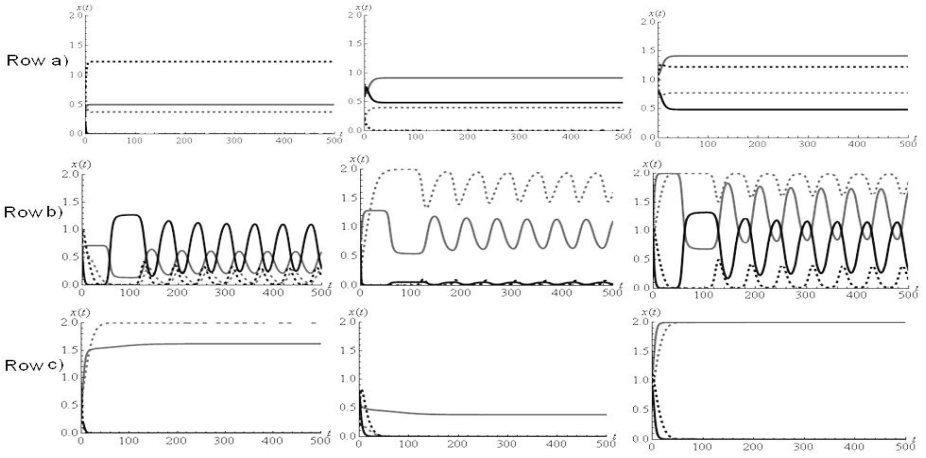


Fig. 3. In all plots the grey, grey-dashed, black, black-dashed curve represents the concentration of s_1, s_2, s_3, s_4 respectively. In rows **a)**, **b**) and **c**) the left plot shows the concentration of molecules in the inner compartment, the middle plot shows the concentration of the outer compartment and the right plot shows the concentration of the sum of inner and outer concentrations (the total concentration in the P system considered as a whole). The plots in row **a**) show the P system when the exchange reaction rates are switched to zero (compartments do not interact with each other). We see that the inner compartment tends to O_7 and the outer compartment to O_8, and thus the P system considered as a whole tends to O_9. The plots in row **b**) show the same system of plot row **a)**, but permitting exchange of molecules through the membranes. It is observed an oscillatory regime in which O_9 is maintained in both compartments, because the exchanged molecules help to the long-term stability of the system. This oscillatory regime is not possible in a single membrane system. The plots in row **c**) show again the same system as in **a**) but with different exchange-reaction rates. This time the exchange reactions lead in asymptotic regime to the non-reactive organization O_6.

Example 5. In this example we show how feasibility and membranes are complementary concepts to understand the asymptotic behavior in biological systems.

We are going to define a P system of two membranes, one internal and the other external, such that there are an inner and an outer compartment. Both compartments have the same reaction rules (we are going to use the reactions of Example 4), and we are going to allow exchange of molecules through the internal membrane (from the inner to outer compartment and vice versa). See Figure 2 (Right) for an illustration.

We allow that reactions are fired with different reaction rates in the inner and outer compartment (this assumption is theoretically plausible for example in charged membranes). This fact makes possible the *coexistence* of the two incompatible organizations O_7 and O_8 of Example 4 (one organization in each membrane), and thus when considering the P system as a whole, we have that the organization O_9 increases its rates-region of feasibility because of the exchange

reactions. Furthermore, for some exchange rates the resulting behavior is an asymptotic stability of O_9 in each compartment, but in an oscillatory regime (if the membrane is destroyed this equilibrium is broken). Finally, for certain exchange rates, the asymptotic behavior of a feasible and reactive organization is a non-reactive organization, this implies that the creation/destruction of membranes can also break the stability of feasible organizations. See Figure 3 for simlations showing this phenomena.

5 Conclusions

We have shown that, given a kinetic law of mass-action type, feasibility can be computed for each organization, i.e. whether it has a corresponding fixed point in the ODE (1) or not. This complements the fixed point theorem (Theorem 1 in [4]) and refines the information organization theory can give about a systems dynamics. We presented an intuitive way to build unfeasible organizations (cf. Example 3). We have shown that the unfeasibility can emerge from combining reactive as well as non-reactive feasible organizations. This means the phenomena of unfeasibility are potentially present in any reaction network. Thus, terms like feasibility or incompatibility of organizations can give rise to formalizations of various ecological-like phenomena in nature, e.g., competition, symbiosis, depredation, etc. (cf. Example 2, 4 and 5). We succesfully applied chemical organization theory to P systems: We have shown that in a P system creation and destruction of membranes allow for the occurrence of unfeasible organizations as well as destruction of feasible organizations, thus enriching the set of possible behaviors of a chemical reaction system.

The interface of Chemical organization theory and P systems provides a wide field for further research promising new analysis techniques for reaction networks. E.g., the concept of compatibility of organizations informally introduced in this paper could be further developed and applied to the above mentioned ecological-like concepts. Future work should also be concerned with the relation between different kinetic laws and feasibility of organizations. Furthermore it should be analyzed how complicated it is to determine feasibility automatically since methods from linear algebra do not suffice.

Acknowledgments. The authors would like to thank Peter Kreyßig and Thomas Hintze for their helpful advices.

References

1. National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov
2. Centler, F., Dittrich, P.: Chemical Organizations in Atmospheric Photochemistries: A New Method to Analyze Chemical Reaction Networks. Planet. Space Sci. 55, 413–428 (2007)
3. Centler, F., Kaleta, C., Speroni di Fenizio, P., Dittrich, P.: Computing Chemical Organizations in Biological Networks. Bioinformatics 24, 1611–1618 (2008)

4. Dittrich, P., Speroni di Fenizio, P.: Chemical Organization Theory. Bull. Math. Biol. 69, 1199–1231 (2007)
5. Feinberg, M., Horn, F.J.M.: Dynamics of Open Chemical Systems and the Algebraic Structure of the Underlying Reaction Network. Chem. Eng. Sci. 29, 775–787 (1974)
6. Fontana, W., Buss, L.W.: The Arrival of the Fittest: Toward a Theory of Biological Organization. Bull. Math. Biol. 56, 1–64 (1994)
7. Luhmann, N.: Soziale Systeme. Suhrkamp, Frankfurt a.M (1984)
8. Matsumaru, N., Centler, F., di Fenizio, P.S., Dittrich, P.: Chemical Organization Theory as a Theoretical Base for Chemical Computing. Int. Jour. on Unconventional Computing 3, 285–309 (2007)
9. Matsumaru, N., Centler, F., di Fenizio, P.S., Dittrich, P.: Chemical Organization Theory Applied to Virus Dynamics. Information Technology 48, 154–160 (2006)
10. Murray, J.D.: Mathematical Biology I: An Introduction. Springer, New York (2002)
11. Peter, S.: Chemische Organisationen und kontinuierliche Dynamik. Diploma Thesis (2008)
12. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, University of Bonn, Bonn (1962)
13. Schilling, C.H., Schuster, S., Palsson, B.O., Heinrich, R.: Metabolic Pathway Analysis: Basic Concepts and Scientific Applications in the Post-genomic Era. Biotechnol. Prog. 15, 296–303 (1999)
14. Schuster, S., Dandekar, T., Fell, D.A.: Detection of Elementary Flux Modes in Biochemical Networks: A Promising Tool for Pathway Analysis and Metabolic Engineering. Trends Biotechnol. 17, 53–60 (1999)
15. Sensse, A.: Convex and Toric Geometry to Analyze Complex Dynamics in Chemical Reaction Systems. Ph.D. thesis, Otto-von-Guericke University Magdeburg, Magdeburg (2005)
16. Sensse, A., Eiswirth, M.: Feedback Loops for Chaos in Activator-inhibitor Systems. Jour. Chem. Phys. 122, 044516–044516-9 (2005)
17. Sensse, A., Hauser, M.J.B., Eiswirth, M.: Feedback Loops for Shilnikov Chaos: The Peroxidase-oxidase Reaction, Jour. Chem. Phys. 125, 014901–014901-12 (2006)
18. Strogatz, S.H.: Nonlinear Dynamics and Chaos. Westview Press, Cambridge (2000)
19. Paun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
20. Bernardini, F., Manca, V.: Dynamical aspects of P systems. Biosystems 70, 85–93 (2003)

P Systems with Elementary Active Membranes: Beyond NP and coNP

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron

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

Abstract. We prove that a uniform family of P systems with active membranes, where division rules only operate on elementary membranes and dissolution rules are avoided, can be used to solve the following **PP**-complete decision problem in polynomial time: given a Boolean formula of m variables in 3CNF, do at least $\sqrt{2^m}$ among the 2^m possible truth assignments satisfy it? As a consequence, the inclusion $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ holds: this provides an improved lower bound on the class of languages decidable by this kind of P systems.

1 Introduction

P systems with active membranes [11] are a variant of P systems where a particularly important role in the computation is performed by the membranes themselves: they possess an electrical charge that can inhibit or activate the rules that govern the evolution of the system, and they can also increase exponentially in number via division rules. The latter feature makes them extremely efficient from a computational complexity standpoint: using exponentially many membranes that evolve in parallel, they can be used to solve **PSPACE**-complete problems [12,2] in polynomial time.

When the ability of dividing membranes is limited, the efficiency apparently decreases. The so-called Milano theorem [14] tells us that no **NP**-complete problem can be solved in polynomial time without using division rules, unless $\mathbf{P} = \mathbf{NP}$ holds.

On the other hand, the computing power of polynomial-time P systems with division rules operating only on *elementary* membranes (that is, membranes not containing other membranes) has not been yet characterised precisely. It is a known fact that elementary division rules suffice to efficiently solve **NP**-complete problems (and, due to closure under complement, also **coNP**-complete ones). This result dates back to 2000 in the semi-uniform case [14], where each input is mapped to a specific P system solving the problem for that particular input, and to 2003 in the uniform case [9], where a single P system solves the problem for all inputs of the same size. In terms of complexity classes, this is written $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})} \subseteq \mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{n})}$, where the star denotes

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 338–347, 2010.

semi-uniformity. Since these results do not require membrane dissolution rules, we also have the (possibly stronger) inclusion $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$; the systems of type $\mathcal{AM}(-\mathrm{d},-\mathrm{n})$ are sometimes called *P systems with restricted elementary active membranes* [2].

No significant improvement on the $\mathbf{NP} \cup \mathbf{coNP}$ lower bound for the complexity classes $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ and $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$, or the corresponding semi-uniform classes, has been found since then, although a **PSPACE** upper bound was proved in 2007 [13].

In 2008, Alhazov et al. [1] proved that P systems with elementary active membranes can be used to solve **PP**-complete problems, but their result is not directly related to $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$, since it requires either cooperative evolution rules, a very strong feature which is not a part of standard P systems with active membranes, or post-processing data of exponential size (when expressed in unary).

The complexity class **PP** appears to be larger than **NP**, since it contains **NP** as a subset and it is closed under complement: thus $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PP}$. In this paper we prove that a **PP**-complete problem (and, as a consequence, the totality of problems in **PP**) can indeed be solved in polynomial time using standard P systems with restricted elementary active membranes.

2 Definitions

We begin by recalling the definition of P systems with restricted elementary active membranes.

Definition 1. *A* P system with restricted elementary active membranes [2]*, in symbols* $\mathcal{AM}(-\mathrm{d},-\mathrm{n})$*, of the initial degree* $d \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$$

where:

- Γ *is a finite alphabet of symbols, also called* objects*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted unordered tree) consisting of* d *membranes enumerated by* $1, \ldots, d$*; furthermore, each membrane is labelled by an element of* Λ*, not necessarily in a one-to-one way;*
- $w_1, \ldots, w_d$ *are strings over* Γ*, describing the multisets of objects placed in the* d *initial regions of* μ*;*
- R *is a finite set of rules.*

Each membrane possesses a further attribute, named *polarization* or *electrical charge*, which is either neutral (represented by 0), positive (+) or negative (−) and it is assumed to be initially neutral.

The rules are of the following kinds:

- *Object evolution rules*, of the form $[a \to w]_h^\alpha$
  They can be applied inside a membrane 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 multiset w).
- *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$
  They can be applied to a membrane labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β.
- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β.
- *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\, [c]_h^\gamma$
  They can be applied to a membrane labelled by h, having charge α, containing an occurrence of the object a but having no other membrane inside; the membrane is divided into two membranes having label h and charge β and γ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes.

A *configuration* in a P system with active membranes is described by its current membrane structure, together with its charges and the multisets of objects contained in its regions. The *initial configuration* is given by μ, all membranes having charge 0 and the initial contents of the membranes being $w_1, \ldots, w_d$. A *computation step* changes the current configuration according to the following principles:

- Each object and each membrane can be subject to only one rule during a computation step.
- The rules are applied in a *maximally parallel way*: each object which appears on the left-hand side of applicable evolution, communication, or elementary division rules must be subject to exactly one of them; the same holds for each membrane which can be involved in a communication or division rule. The only objects and membranes which remain unchanged are those associated with no rule, or with rules that are not applicable in that particular step (due to the charge of the membrane).
- When more than one rule can be applied to an object or membrane, the actual rule to be applied is chosen nondeterministically; hence, in general, multiple configurations can be reached from the current one.
- When division rules are applied to a membrane, the multiset of objects to be copied is the one resulting *after* all evolution rules have been applied.
- The skin membrane cannot be divided. Furthermore, every object which is sent out from the skin membrane cannot be brought in again.

A *halting computation* $\boldsymbol{\mathcal{C}}$ of a P system Π is a finite sequence of configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration of Π, every $\mathcal{C}_{i+1}$ can be reached from $\mathcal{C}_i$ according to the principles just described, and no further configuration can be reached from $\mathcal{C}_k$ (i.e., no rule can be applied). P systems might also perform *non-halting* computations; in this case, we have infinite sequences $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ of successive configurations.

We can use families of P systems with active membranes as language recognisers, thus allowing us to solve decision problems.

Definition 2. *A* recogniser P system with active membranes Π *has an alphabet containing two distinguished objects* YES *and* NO*, used to signal acceptance and rejection respectively; every computation of* Π *is halting and exactly one object among* YES, NO *is sent out from the skin membrane during each computation.*

In what follows we will only consider *confluent* recogniser P systems with active membranes, in which all computations starting from the same initial configuration agree on the result.

Definition 3. *Let* $L \subseteq \Sigma^\star$ *be a language and let* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *be a family of recogniser P systems. We say that* $\boldsymbol{\Pi}$ decides L*, in symbols* $L(\boldsymbol{\Pi}) = L$*, when for each* $x \in \Sigma^\star$*, the result of* Π_x *is acceptance iff* $x \in L$*.*

Usually some uniformity condition, inspired by those applied to families of Boolean circuits, is imposed on families of P systems. Two different notions of uniformity have been considered in the literature; they are defined as follows.

Definition 4. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* semi-uniform *when the mapping* $x \mapsto \Pi_x$ *can be computed in polynomial time, with respect to* $|x|$*, by a deterministic Turing machine.*

Definition 5. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* uniform *when there exist two polynomial-time deterministic Turing machines* M_1 *and* M_2 *such that, for each* $n \in \mathbb{N}$ *and each* $x \in \Sigma^n$

- M_1*, on input* 1^n *(the unary representation of the length of* x*), outputs the description of a P system* Π_n *with a distinguished input membrane;*
- M_2*, on input* x*, outputs a multiset* w_x *(an encoding of* x*);*
- Π_x *is* Π_n *with* w_x *added to the multiset located inside its input membrane.*

In other words, the P system Π_x *associated with string* x *consists of two parts; one of them,* Π_n*, is common for all strings of length* $|x| = n$ *(in particular, the membrane structure and the set of rules fall into this category), and the other (the input multiset* w_x *for* Π_n*) is specific to* x*. The two parts are constructed independently and, only as the last step,* w_x *is inserted in* Π_n*.*

Time complexity classes for P systems [9] are defined as usual, by restricting the amount of time available for deciding a language. By $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ (resp., $\mathbf{PMC}^\star_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$) we denote the class of languages which can be decided by

uniform (resp., semi-uniform) families $\boldsymbol{\Pi}$ of confluent P systems with restricted elementary active membranes where each computation of $\Pi_x \in \boldsymbol{\Pi}$ halts in polynomial time with respect to $|x|$. These classes are known to be closed under complement and polynomial-time reductions. Since uniformity is a special case of semi-uniformity, the inclusion $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} \subseteq \mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ holds; we only consider uniform families in the rest of the paper.

The complexity class **PP** (Probabilistic **P**) was first introduced to characterise those decision problems which can be solved efficiently by a probabilistic Turing machine, whose probability of error on every input is strictly less than 1/2 [6]. An equivalent definition of **PP** is usually given in terms of nondeterministic Turing machines by altering the notion of acceptance [8].

Definition 6. *The complexity class* **PP** *consists of all languages $L \subseteq \Sigma^{\star}$ which can be decided in polynomial time by a nondeterministic Turing machine N with the following acceptance criterion: N accepts $x \in \Sigma^{\star}$ iff more than half of the computations of N on input x are accepting.*

3 Solving a PP-Complete Problem

One of the standard **PP**-complete problems is Majority-SAT [4,8]: given a Boolean formula φ of m variables in conjunctive normal form, determine whether more than half of the 2^m possible truth assignments satisfy it. However, it is not easy to provide a polynomial-time *uniform* solution for this problem, since the clauses of φ may contain any number of literals between 1 and $2m$. The usual solution in membrane computing is to require the input formula to have exactly three different literals per clause (see, e.g., [10]). Unfortunately, the resulting decision problem Majority-3SAT is not known to be **PP**-complete. In particular, the standard reduction from SAT to 3SAT [5] is not applicable here, as it requires the addition of "dummy" variables, which increase the number of possible assignments without necessarily increasing the number of satisfying ones: this can decrease the ratio of satisfying assignments over total assignments from above 1/2 to a value less than or equal to this threshold.

There is, however, yet another slight variation of the problem that is suitable for our purposes.

Definition 7. Sqrt-3SAT[1] *is the following decision problem: given a Boolean formula of m variables in 3CNF, determine whether the number of truth assignments satisfying it is at least $\sqrt{2^m}$.*

The problem Sqrt-3SAT is known to be **PP**-complete [3], and it is very close in spirit to Majority-3SAT. Our solution to this problem follows the canon for **NP**-complete problems in membrane computing [11,14], but with an additional intermediate phase (numbered 3 in the following algorithm).

[1] This problem is denoted by $\#3\mathrm{SAT}(\geq 2^{m/2})$ in the original paper [3].

Algorithm 1. *Solving* SQRT-3SAT *on input φ, a 3CNF formula of m variables.*

1. *Generate 2^m membranes using elementary division, each one containing a different truth assignment to the variables occurring in φ.*
2. *Evaluate φ under the 2^m assignments, in parallel, and send out from each membrane an object t whenever the formula is satisfied by the corresponding assignment.*
3. Erase $\lceil\sqrt{2^m}\rceil - 1$ instances of t (or all of them, if less than $\lceil\sqrt{2^m}\rceil - 1$ occur).
4. *Output* YES *if at least one instance of t remains; otherwise, output* NO.

Notice that, by removing Phase 3, we obtain the standard membrane computing algorithm for SAT. The additional phase was first proposed by Alhazov et al. [1] for checking the value of the permanent of a matrix, but the authors used cooperative object evolution rules, that are not part of standard P systems with active membranes. In Section 3.2 we show how to implement this phase using elementary division and communication rules, together with all the other steps of Algorithm 1.

3.1 Encoding of Formulae

Formulae in 3CNF are easy to encode as binary strings [7,10]. Given m variables, only $8\binom{m}{3}$ clauses without repeated variables exist: we have $\binom{m}{3}$ sets of three out of m variables, and each one of them can be either positive or negated. Once an easily-computable enumeration of the clauses has been fixed (e.g., under a lexicographic order, the i-th clause can be computed from i in polynomial time) a formula φ can be represented by a string $\langle\varphi\rangle$ of $n = 8\binom{m}{3}$ bits, where the i-th bit is set iff the i-th clause occurs in φ.

Under this encoding, a string in $\{0,1\}^n$ is a valid formula iff $n = 8\binom{m}{3}$ for some integer $m \geq 3$. The number of variables m can be easily recovered in polynomial time, given n in unary notation, by finding the unique positive integer root of the polynomial $p(m) = 8\binom{m}{3} - n = \frac{4}{3}m^3 - 4m^2 + \frac{8}{3}m - n$. If no such root exists, we can deduce that the input is not well-formed with respect to our encoding.

3.2 Solution to Sqrt-3SAT

The implementation of Algorithm 1 is a uniform variant of the solution described by Zandron et al. [14]. To all strings $x \in \{0,1\}^n$ with $n = 8\binom{m}{3}$, representing Boolean formulae φ of m variables, we associate a P system with restricted elementary active membranes Π_n. The initial configuration of Π_n (excluding the input multiset) is the following one:

$$\mathcal{C}_0 = \left[q_0 r_0 \;\; [p_0 x_1 x_2 \cdots x_m]_1^0 \;\; [b_{i_1}]_2^0 [b_{i_2}]_2^0 \cdots [b_{i_h}]_2^0\right]_0^0$$

Here the objects $x_1, \ldots, x_m$ represent the variables of φ, while p_0, q_0, and r_0 are objects used to implement three timers, counting from zero.

The number of membranes having label 2 and their contents are determined as follows. Let $k = \lceil\sqrt{2^m}\rceil - 1$, and consider the binary representation of k:

for each $i = 0, \ldots, \lfloor \log k \rfloor$, if the i-th least significant bit of k is 1, then we add to $\mathcal{C}_0$ a copy of membrane 2, containing the single object b_i; otherwise we add nothing. In other words, h and $i_1 < i_2 < \cdots < i_h$ are the unique integers such that $k = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_h}$. Clearly h is bounded by k, which is in turn bounded by $\frac{m}{2}$; hence the configuration $\mathcal{C}_0$ can be costructed in polynomial time with respect to n.

The input multiset, obtained from $\langle \varphi \rangle$, is placed inside membrane 1, and contains all the objects c_i such that the i-th clause does *not* occur in φ.

For instance, suppose $m = 3$, hence $n = 8\binom{3}{3} = 8$. The eight (up to reordering of literals) clauses over three variables x_1, x_2, x_3 can be enumerated as

$$\begin{array}{llll} x_1 \vee x_2 \vee x_3 & x_1 \vee x_2 \vee \neg x_3 & x_1 \vee \neg x_2 \vee x_3 & x_1 \vee \neg x_2 \vee \neg x_3 \\ \neg x_1 \vee x_2 \vee x_3 & \neg x_1 \vee x_2 \vee \neg x_3 & \neg x_1 \vee \neg x_2 \vee x_3 & \neg x_1 \vee \neg x_2 \vee \neg x_3 \end{array}$$

and the formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ is then encoded as $\langle \varphi \rangle = 0010\ 0110$. The corresponding input multiset is $c_1 c_2 c_4 c_5 c_8$.

Starting from the initial configuration, including the input multiset, the computation proceeds as follows.

Phase 1 (Generate). Each variable object x_i is used to divide membrane 1, and is replaced by a "true" object t_i on one side, and by a "false" object f_i on the other, denoting the two possible truth values that can be assigned to variable x_i. The corresponding rules are

$$[x_i]_1^0 \to [t_i]_1^0 [f_i]_1^0 \qquad \text{for } 1 \leq i \leq m.$$

While the membranes having label 1 divide, thus generating 2^m copies (each one containing a different assignment), the timer p_0 is incremented, one step at a time, up to m:

$$[p_j \to p_{j+1}]_1^0 \qquad \text{for } 0 \leq j \leq m-1.$$

The object p_m is then sent out of membrane 1, while changing the charge of the membrane to positive, using the rule $[p_m]_1^0 \to [\,]_1^+ p_m$.

The object p_m is immediately brought back in (and renamed to u_0) via $p_m\ [\,]_1^+ \to [u_0]_1^+$. Simultaneously, each object t_i and f_i is replaced by a set of objects denoting the clauses that are satisfied when the variable x_i is true or false, respectively, i.e.:

$$\begin{array}{ll} [t_i \to c_{i_1} \cdots c_{i_s}]_1^+ & \text{for } 1 \leq i \leq m \text{ and } x_i \text{ occurs in clauses } i_1, \ldots, i_s; \\ [f_i \to c_{i_1} \cdots c_{i_s}]_1^+ & \text{for } 1 \leq i \leq m \text{ and } \overline{x}_i \text{ occurs in clauses } i_1, \ldots, i_s. \end{array}$$

Notice that computing these sets of clauses does *not* require the input formula φ, but only its size n (this is consistent with a uniform construction). The clause-objects c_j are produced in the $(m+2)$-th step.

While all these events described above occur inside the membranes labelled by 1, we also divide the membranes with label 2 until we have $\lceil \sqrt{2^m} \rceil - 1$ copies. Indeed, each object b_i is used to create 2^i copies, according to the following rules:

$$[b_i]_2^0 \to [b_{i-1}]_2^0 [b_{i-1}]_2^0 \qquad \text{for } 1 \leq i \leq \lfloor \log k \rfloor.$$

Producing all copies of membrane 2 requires a number of steps bounded by

$$\lfloor \log k \rfloor = \lfloor \log(\lceil \sqrt{2^m} \rceil - 1) \rfloor \leq \log \lceil \sqrt{2^m} \rceil \leq \frac{m}{2} + 1 \leq m + 2.$$

Hence, Phase 1 requires a total of $m + 2$ steps.

Phase 2 (Evaluate). In this phase, the object u_j inside each copy of membrane 1 behaves as a counter for the number of satisfied clauses, and initially it has the value u_0.

Now consider the contents of the membranes having label 1. If the i-th clause occurs in φ *and* it is satisfied by the truth assignment corresponding to the particular copy of membrane 1 under consideration, then one or more instances of object c_i have been generated in Phase 1. If this clause does *not* occur in φ, then the object c_i has been placed in membrane 1 as part of the input multiset: the clause is then considered to be satisfied[2]. Finally, if this clause *does* occur in φ but it is not satisfied, then no instance of c_i occurs inside membrane 1.

We find out whether the clauses are satisfied, one by one in the order established in Section 3.1, by checking whether an instance of the object c_1 occurs, then decrementing the subscript of all the other objects c_j by one; this procedure is repeated until an unsatisfied clause is found, or all of them are found to be satisfied.

If c_1 does indeed occur, then it is sent out and changes the charge of 1 to negative, using the rule $[c_1]_1^+ \to [\ ]_1^-\ c_1$. While membrane 1 is negative, the other subscripts are decremented:

$$[c_j \to c_{j-1}]_1^- \qquad \text{for } 1 \leq j \leq n.$$

Simultaneously, u_j increments its subscript via

$$[u_j \to u_{j+1}]_1^- \qquad \text{for } 0 \leq j < n,$$

and c_1 re-enters membrane 1 (not necessarily the same instance of membrane 1, but any negatively charged one) as the "junk" object #, and sets its charge back to positive via $c_1\ [\ ]_1^- \to [\#]_1^+$.

Phase 2 now restarts, with all clause-objects having their subscript decremented by one. If one of the objects c_j is missing for some $j = 1, \ldots, n$, then the computation in that copy of membrane 1 halts prematurely, and u_n is never reached. On the other hand, if all objects $c_1, \ldots, c_n$ exist inside a certain copy of membrane 1, the object u_n is reached in $2n$ steps: we can then conclude that the formula is completely satisfied, and send out a t object to signal it, using the rule $[u_n]_1^+ \to [\ ]_1^+\ t$. Notice that all t objects are sent out simultaneously from all copies of membrane 1.

The total number of steps required for Phase 2 is $2n + 1 = 16\binom{m}{3} + 1$.

Phase 3 (Erase). When the instances of object t reach the skin membrane, labelled by 0, each copy of membrane 2 absorbs one of them, if any is available,

[2] This is consistent with the "true" value being the identity of conjunction.

using the communication rule $t\,[\,]_2^0 \to [\#]_2^+$. After this computation step, one or more copies of t remain inside membrane 0 iff the number of instances of t was at least $\sqrt{2^m}$, that is, iff φ is a positive instance of Sqrt-3SAT.

Phase 4 (Output). The sequences of objects q_j and r_j, which begin with q_0 and r_0 and whose behaviour we have not described yet, are meant to count the number of steps across Phases 1, 2, and 3, that is, $\ell = (m+2) + \left(16\binom{m}{3} + 1\right) + 1$. This is accomplished by using the following evolution rules:

$$[q_j \to q_{j+1}]_0^0 \qquad \text{for } 0 \leq j \leq \ell;$$
$$[r_j \to r_{j+1}]_0^\alpha \qquad \text{for } 0 \leq j \leq \ell + 2 \text{ and } \alpha \in \{+, 0, -\}.$$

When the subscript of q reaches ℓ, Phase 3 has just finished. This object is sent out in order to change the charge of membrane 0 to positive, using the rule $[q_\ell]_0^0 \to [\,]_0^+ \#$; this enables any remaining instance of t inside membrane 0 to exit and change again the charge of the skin to negative, using the rule $[t]_0^+ \to [\,]_0^- \#$. If no object t exists inside membrane 0, the charge remains positive.

During the next computation step, the subscript of r is $\ell + 2$, and this object is finally sent out, either as YES or NO depending on the charge of membrane 0:

$$[r_{\ell+2}]_0^+ \to [\,]_0^+ \text{ NO} \qquad [r_{\ell+2}]_0^- \to [\,]_0^- \text{ YES}.$$

According to the argument above, the object emerging from membrane 0 corresponds to the correct answer to the problem.

The sizes of the sets of rules (hence, also the size of the alphabet) described in each step of the algorithm are clearly bounded by a polynomial in n and computable efficiently from 1^n; thus, we can conclude that Sqrt-3SAT has a polynomial time uniform solution.

Theorem 1. Sqrt-3SAT $\in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$, *hence* $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ *via polynomial-time reductions.* □

4 Conclusions

We improved one of the earliest results related to P systems with active membranes, namely that elementary division is sufficient to solve **NP**-complete problems in polynomial time, by proving that **PP** problems can also be solved efficiently by the same class of P systems, without the need for dissolution rules. The method is a generalisation of the classic membrane computing algorithm schema for **NP**-complete problems, where all candidate solutions are generated and then tested in parallel.

This result does still not provide, however, a characterisation of the complexity classes $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ and $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ in terms of Turing machines; furthermore, neither the **PP** lower bound, nor the **PSPACE** upper bound are known to be strict. We think that the question is worth further investigation, with the goal of finally establishing whether nonelementary division rules are a redundant feature of P systems with active membranes, or a fundamental one.

References

1. Alhazov, A., Burtseva, L., Cojocaru, S., Rogozhin, Y.: Solving PP-complete and #P-complete problems by P systems with active membranes. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 108–117. Springer, Heidelberg (2009)
2. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundamenta Informaticae 58(2), 67–77 (2003)
3. Bailey, D.D., Dalmau, V., Kolaitis, P.G.: Phase transitions of PP-complete satisfiability problems. Discrete Applied Mathematics 155(12), 1627–1639 (2007)
4. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity I, 2nd edn. Springer, Heidelberg (1995)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York (1979)
6. Gill, J.T.: Computational complexity of probabilistic Turing machines. In: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 91–95 (1974)
7. Lagoudakis, M.G., LaBean, T.H.: 2D DNA self-assembly for satisfiability. In: Winfree, E., Gifford, D.K. (eds.) DNA Based Computers V. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 54, pp. 139–152 (1999)
8. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1993)
9. Pérez-Jiménez, M.J., Romero Jiménez, A., Sancho Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–285 (2003)
10. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. Natural Computing (in press), doi:10.1007/s11047-010-9189-x
11. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
12. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
13. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
14. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K: Proceedings of the Second International Conference, pp. 289–301. Springer, Heidelberg (2001)

Polynomial Complexity Classes in Spiking Neural P Systems

Petr Sosík[1,2], Alfonso Rodríguez-Patón[1], and Lucie Ciencialová[2]

[1] Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain
{psosik,arpaton}@fi.upm.es
[2] Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava, 74601 Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz

Abstract. We study the computational potential of spiking neural (SN) P systems. several intractable problems have been proven to be solvable by these systems in polynomial or even constant time. We study first their formal aspects such as the input encoding, halting versus spiking, and descriptional complexity. Then we establish a formal platform for complexity classes of uniform families of confluent recognizer SN P systems. Finally, we present results characterizing the computational power of several variants of confluent SN P systems, characterized by classes ranging from **P** to **PSPACE**.

1 Introduction

Spiking neural P system (abbreviated as SN P system) introduced in [3] is an abstract computing model inspired by the theory of membrane computing, on one hand, and spiking neural networks, on the other hand. The computational power of SN P system has been extensively studied and several intractable problems such as Subset Sum, SAT, QSAT and others have been shown effectively solvable by SN P systems under various conditions.

We intend to establish a platform allowing to characterize the computational power of SN P systems more precisely. Computational complexity theory provides basic tools for this task. Necessary formal prerequisites are given in Section 3, including the input encoding of SN P systems, requirements for halting and providing an output, and the size of the description of an SN P system. These postulates allow to define polynomially uniform families of *confluent* SN P systems in Section 4 and to study their properties. Then we focus on uniform families of SN P systems. We show the closure of their polynomial time-restricted complexity classes under complement and polynomial time reduction. Finally we provide a characterization or limitation of some variants of uniform families of recognizer SN P system in Section 5. We also mention the differences between unary and binary encoding and study their influence on the presented results. We

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 348–360, 2010.

show that some restricted variants of regular expressions (including the single star normal form) characterize the class **P**, while in general their computational power lies between classes **NP**, **co-NP** and **PSPACE**.

2 Prerequisites

We assume the reader to be familiar with basic language and automata theory, as well as with elements of the computational complexity theory. We also refer to [10] for an up-to-date information about membrane computing.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The length of a string $x \in V^*$ is denoted by $|x|$. Next we recall the definition of regular expression to fix the notation.

Definition 1. *For a finite alphabet V : (i) λ and each $a \in V$ are regular expressions, (ii) if E_1, E_2 are regular expressions over V, then also $(E_1) \cup (E_2), (E_1) \cdot (E_2)$, and $(E_1)^*$ are regular expressions over V, and (iii) nothing else is a regular expression over V.*

The catenation operator $\cdot$ and non-necessary parentheses may be omitted when writing a regular expression. With each expression E we associate its language $L(E)$ defined in a usual way. We call two expressions E_1 and E_2 *equivalent* if $L(E_1) = L(E_2)$.

Definition 2 ([1]). *We say that a regular expression $E = E_1 \cup \ldots \cup E_n$ (where each E_i contains only $\cdot$ and $*$ operators) is in* single-star normal form *(SSNF) if $\forall i \in \{1, \ldots, n\}$, E_i has at most one occurrence of $*$.*

Lemma 1 ([1]). *Every regular expression over one-letter alphabet can be transformed into an equivalent single-star normal form.*

This transformation, however, might require an exponential time and the size of the resulting expression can be exponential with respect to the size of the original expression.

3 Spiking Neural P Systems

A *spiking neural membrane system* of degree $m \geq 1$ is a construct of the form

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

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

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

where:

a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
b) R_i is a finite set of *rules* of the following two forms:
   (1) $E/a^c \to a;d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
   (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \to a;d$ of type (1) from R_i, we have $a^s \notin L(E)$;

3. $syn \subseteq \{1,2,\ldots,m\} \times \{1,2,\ldots,m\}$ with $(i,i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
4. $in, out \in \{1,2,\ldots,m\}$ indicate the *input neuron* (resp., *output neuron*).

The rules of type (1) are *firing* (we also say *spiking*) *rules*. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a;d$ can be applied. The application of this rule means consuming (removing) c spikes (thus only $k-c$ remain in σ_i), the neuron is fired, and it produces a spike after d time units (a global clock is assumed, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed* so that it cannot receive new spikes. If a neuron has a synapse to a closed neuron and tries to send a spike along it, then the spike is lost. In the step $t+d$, the neuron spikes and becomes again open so that it can receive spikes.

The rules of type (2) are *forgetting* rules; they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \to a;d$ of type (1) has $E = a^c$, then we will write it in the following simplified form: $a^c \to a;d$.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used. Since two firing rules, $E_1/a^{c_1} \to a;d_1$ and $E_2/a^{c_2} \to a;d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and in that case, only one of them is chosen non-deterministically. Note that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

A *configuration* of the system is described by the numbers of spikes $n_1, n_2, \ldots, n_m$ present in each neuron, and the number of steps remaining until the neuron becomes open (if the neuron is already open, the number is 0). The initial configuration of the system is described by the initial numbers of spikes present in each neuron, with all neurons being open.

Using the rules described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation (halting or not) we associate a *spike train*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0. We refer the reader to [12] for more details.

3.1 Unary versus Binary Input/Output

Original works on SN P systems, e.g., [3,11] focused on SN P systems working in the generating mode. The output value(s) was represented as a time interval between two spikes of a designated neuron, i.e., in *unary.* A similar convention was later adopted also for input [6].

Unary input/output encoding
An input/output sequence of positive natural numbers $n_1, \ldots, n_k$ is represented as a spike train $10^{n_1-1}10^{n_2-1}1 \ldots 10^{n_k-1}1$.

However, it is known that standard SN P systems can simulate logic gates with unbounded fan-in in a unit time [2] and, hence, also arbitrary logic circuits in linear time. The unary input/output convention would decrease their computational power *exponentially* in many cases. Therefore we propose a binary encoding.

Binary input/output encoding
An input/output to an SN P system is a binary spike train received/emitted by a designated input/output neuron.

Although certain aspects of the binary encoding can be also found problematic, we use the binary encoding in the rest of the paper. Obviously, to switch from binary to unary encoding, one needs a time exponential with respect to the size of the original binary string unless an extended SN P system with maximal parallelism is used [8].

3.2 Recognizer SN P Systems

In this subsection we define SN P systems solving decision problems. Let us call *decision problem* a pair $X = (I_X, \theta_X)$ where I_X is a language over a finite alphabet (whose elements are called instances) and θ_X is a total boolean function over I_X. The following convention was suggested by some authors: an SN P systems solving an instance $w \in I_X$ would halt if and only if $\theta_X(w) = 1$. Such an SN P system is called *accepting* in [6]. However, this convention was rarely implemented. Instead, many authors demonstrated SN P systems which always halt and the output neuron spikes if and only if $\theta_X(w) = 1$, see [2,5,6,7,8] and others. We add that this convention is more compatible with definitions of standard complexity classes and also with standard construction of families of recognizer P systems [13]. Hence we suggest the following definition:

Definition 3. *A* recognizer SN P system *satisfies the following conditions: all computations are halting, and the output neuron spikes no more than once during each computation. The computation is called* accepting *if the output neuron spikes exactly once, otherwise it is* rejecting.

Observe that the definition is compatible with the variant when the system is asked to spike *at least once* in the case of accepting computation. To any such SN P system we can add another neuron connected to the original output, with

two initial spikes and the rules $a \to \lambda$ and $a^3 \to a; 0$ which emits only the first spike of those received.

Actually, the difference between the halting and spiking convention is not so great. The following result is demonstrated in [6].

Lemma 2. *Given a system Π with standard or extended rules, with or without delays, we can construct a system Π' with rules of the same kinds as those of Π which spikes if and only if Π halts.*

Note that Π' does not have to halt if Π does not halt. In the other direction, we can extend results in [6] using a similar technique:

Lemma 3. *Given a system Π with standard or extended rules, with or without delays, we can construct a system Π' with rules of the same kinds as those of Π which halts if and only if Π spikes.*

Proof. Consider an SN P system Π (possibly with extended rules), and let σ_{out} be its output neuron. We "triple" this system by:

- tripling the number of spikes present in the initial configuration in each neuron,
- replacing each rule $E/a^c \to a^p; d$ with $3E/a^{3c} \to a^p; d$, where $3E$ is a regular expression for the set $\{www \mid w \in L(E)\}$,
- tripling each neuron σ_c : adding two identical neurons $\sigma_{c'}$, $\sigma_{c''}$ and adding new synapses: if σ_c had originally a synapse to a neuron γ, now each of σ_c, $\sigma_{c'}$ and $\sigma_{c''}$ will have synapses to each of γ, γ' and γ''.

Let us denote by 3Π the obtained system. In this way the behavior of the system is not changed, in the sense that each neuron in 3Π spikes if and only if its original version spikes in Π. If a neuron σ_c in Π contains n spikes in a certain moment, then each of σ_c, $\sigma_{c'}$ and $\sigma_{c''}$ in 3Π contains $3n$ spikes.

Let d_{max} denote the maximal delay in any neuron of Π. Let us construct a module with an incoming synapse from σ_{out} which produces $d_{max}+1$ consecutive spikes when obtaining a spike from σ_{out}, see Fig. 1. The neuron σ_{halt} will have

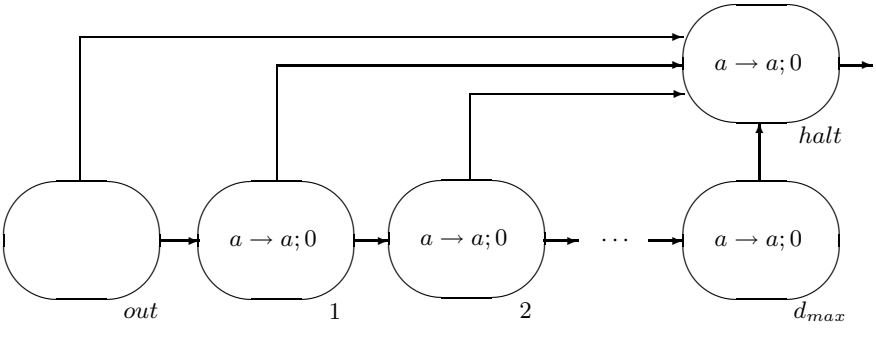


Fig. 1. A module emitting $d_{max} + 1$ consecutive spikes after the output neuron spikes

an outgoing synapse to each neuron of 3Π. Even if some of these neurons may be closed due to their refractory period, this period would end after $\leq d_{max}$ steps. Hence, during $d_{max}+1$ steps when the neuron σ_{halt} spikes, each neuron of 3Π receives a spike, its accumulated number spikes increases to $3n+1$, $n \geq 0$, and none of its rules can be applied. However, more spikes from σ_{halt} may come. Hence, we add to each neuron of 3Π the rule $(aaa)^*aa/a \rightarrow \lambda$ to remove additional spikes. In this way, each neuron will keep $3n+1$ spikes and the system eventually halts after $d_{max}+2$ steps.

Finally, let us add to 3Π a "perpetuum mobile" circuit of two mutually interconnected neurons with a single initial spike and the rule $a \rightarrow a;0$ in each, and with an incoming synapse from σ_{halt}. In this way we ensure that 3Π halts *only* if Π spikes.

3.3 Descriptional Complexity and Size of SN P Systems

According to [7], the size of a SN P system Π is based on the number of bits necessary to its full description. Let m be the number of neurons, N be the maximum natural number that appears in the definition of Π, R the maximum number of rules which occur in its neurons, and S the maximum size required by the regular expressions in succinct form that occur in Π. (The succinct form means that an expression a^n is represented just by $\mathcal{O}(\log n)$ bits.) Then the total size of description of Π is polynomial with respect to m, R, S and $\log N$.

Some authors [6,8,12] distinguish between the size of an SN P system and the size of its description. They point out that the initial number of spikes in neurons or the length of (unary) strings in regular expressions can be exponential with respect to the size of description of the system. However, these unary strings and regular expressions can be represented in the *succinct form*, as a binary value, and neither natural nor artificial implementations of spiking neurons represent spikes as physical strings. Hence we assume the size of an SN P system and the size of its description the same.

4 Families of Recognizer SN P Systems

Standard SN P systems were shown to be computationally universal already in the introductory paper [3]. However, as demonstrated in [9], no standard spiking neural P system with a constant number of neurons can simulate Turing machines with less than exponential time and space overheads. This is due to the unary character of its unlimited memory - spikes accumulated in neurons. Therefore, to achieve computational effectiveness in solving problems, many authors have used families of SN P systems such that each member of a family solves only a finite set of instances of a given size. In this section we propose a formal specification for families of SN P systems. All definitions in this section are inspired by [13] which studies families of P systems working with objects.

Definition 4. *A family* $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ *(respectively,* $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$*) of recognizer SN P without input (resp., with input) is* polynomially uniform by Turing machines *if there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *(resp.,* $\Pi(n)$*) from the instance* $w \in I_X$ *(resp., from* $n \in \mathbb{N}$*).*

In the sequel we will denote such a family simply as *uniform.* The selection of a proper member of the family and its input in the case of families of SN P systems with input is done as follows.

Definition 5. *Let* $X = (I_X, \theta_X)$ *be a decision problem, and* $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ *a family of recognizer SN P systems with input membrane. A polynomial encoding of* X *in* $\mathbf{\Pi}$ *is a pair* $(cod; s)$ *of polynomial-time computable functions over* I_X *such that for each instance* $w \in I_X$, $s(w)$ *is a natural number (obtained by means of a reasonable encoding scheme) and* $cod(w)$ *is a binary string – an input of the system* $\Pi(s(w))$.

The common case is that $s(w)$ is the size of w. Since Definition 4 and 5 conform those in [13,14] we can adopt the following result whose proof in [14] is not affected by a different type of P system.

Lemma 4. *Let* X_1, X_2 *be decision problems,* r *a polynomial-time reduction from* X_1 *to* X_2*, and* $(cod; s)$ *a polynomial encoding of* X_2 *in* $\mathbf{\Pi}$*. Then,* $(cod \circ r; s \circ r)$ *is a polynomial encoding of* X_1 *in* $\mathbf{\Pi}$.

Let $\mathcal{R}$ denote an arbitrary type of recognizer SN P systems. The following definitions are inspired by [13] and [6].

Definition 6. *Let* $f : \mathbb{N} \to \mathbb{N}$ *be a constructible function. A decision problem* X *is solvable by a family* $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ *of recognizer SN P systems of type* $\mathcal{R}$ *without input in time bounded by* f*, denoted by* $X \in \mathbf{SN}^*_{\mathcal{R}}(f)$*, if the following holds:*

- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines.*
- *The family* $\mathbf{\Pi}$ *is* f*-bounded with respect to* X*; that is, for each instance* $w \in I_X$*, every computation of* $\Pi(w)$ *performs at most* $f(|w|)$ *steps.*
- *The family* $\mathbf{\Pi}$ *is sound with respect to* X*; that is, for each* $w \in I_X$*, if there exists an accepting computation of* $\Pi(w)$*, then* $\theta_X(w) = 1$.
- *The family* $\mathbf{\Pi}$ *is complete with respect to* X*; that is, for each* $w \in I_X$*, if* $\theta_X(w) = 1$*, then every computation of* $\Pi(w)$ *is an accepting computation.*

Note that the SN P system solving w can be generally nondeterministic, i.e, it may have different possible computations, but with the same result. Such a P system is also called *confluent.*

The family $\mathbf{\Pi}$ is said to provide a *semi-uniform solution* to the problem X. In this case, for each instance of X we have a special P system. Specifically, we denote by

$$\mathbf{PSN}^*_{\mathcal{R}} = \bigcup_{f \text{ polynomial}} \mathbf{SN}^*_{\mathcal{R}}(f)$$

the class of problems to which uniform families of SN P systems of type $\mathcal{R}$ without input provide semi-uniform solution in polynomial time. Analogously we define families which provide uniform solutions solutions to decision problems.

Definition 7. *Let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function. A decision problem X is solvable by a family $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ of recognizer SN P systems of type $\mathcal{R}$ with input in time bounded by f, denoted by $X \in \mathbf{SN}_{\mathcal{R}}(f)$, if the following holds:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *There exists a polynomial encoding (cod, s) of X in Π such that:*
  - *The family $\mathbf{\Pi}$ is f-bounded with respect to X; that is, for each instance $w \in I_X$, every computation of $\Pi(s(w))$ with input $cod(w)$ performs at most $f(|w|)$ steps.*
  - *The family $\mathbf{\Pi}$ is sound with respect to (X, cod, s); that is, for each $w \in I_X$, if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$, then $\theta_X(w) = 1$.*
  - *The family $\mathbf{\Pi}$ is complete with respect to (X, cod, s); that is, for each $w \in I_X$, if $\theta_X(w) = 1$, then every computation of $\Pi(s(w))$ with input $cod(w)$ is an accepting computation.*

The family $\mathbf{\Pi}$ is said to provide a *uniform solution* to the problem X. Again, we denote by

$$\mathbf{PSN}_{\mathcal{R}} = \bigcup_{f \text{ polynomial}} \mathbf{SN}_{\mathcal{R}}(f)$$

the class of problems to which uniform families of SN P systems of type $\mathcal{R}$ with input provide uniform solution in polynomial time. Obviously, for any constructible function f and a class of SN P systems $\mathcal{R}$ we have

$$\mathbf{SN}_{\mathcal{R}}(f) \subseteq \mathbf{SN}^*_{\mathcal{R}}(f) \quad \text{and} \quad \mathbf{PSN}_{\mathcal{R}} \subseteq \mathbf{PSN}^*_{\mathcal{R}}.$$

To describe a specific type $\mathcal{R}$ of SN P systems, we denote:

$-reg$ for systems with regular expressions of the form a^n, $n \geq 1$,
$-del$ for systems without delays,
$ssnf$ for systems with regular expressions in the single-star normal form.

When $\mathcal{R}$ is omitted, the standard definition of SN P systems is assumed.

Theorem 1. *The classes $\mathbf{SN}_{\mathcal{R}}(f)$ and $\mathbf{SN}^*_{\mathcal{R}}(f)$ are closed under the operation of complement, for $\mathcal{R}$ omitted or $\mathcal{R} \in \{-reg, -del, ssnf\}$.*

Proof. We show that for each confluent SN P system Π there is a system Π' whose computation is accepting if and only if the computation of Π is rejecting. Our presentation is set as general as possible, taking into the account also extended SN P systems, even if they are not mentioned in the statement of the theorem.

Assume the construction described in Section 4, Fig. 6 in [6]. It presents a module which, when added to any SN P system Π, emits a spike only after the system Π halts. This module contains a set of rules $a^k \to a; 0$ for all $k \in K$, where K is the set constructed as follows. For each neuron σ_i of Π, $1 \leq i \leq n$ (where n is the degree of Π) denote

$$P_i = \bigcup_{1 \leq i \leq n} \{p \,|\, E/a^c \to a^p; d \text{ is a rule of } \sigma_i\},$$

and

$$K = \left\{ \sum_{i=1}^{n} p_i \,|\, p_i \in P_i \right\} - \{0\}. \tag{1}$$

Hence K contains sums of all possible n-tuples containing one element of each P_i, hence the number of these n-tuples may be exponential with respect to n. However, in such a case many of these sums will be equal. Let

$$p_{\max} = \max\{p \,|\, E/a^c \to a^p; d \text{ is a rule of } \sigma_i\},$$

then each sum on the right-hand side of (1) will be bounded by $np_{\max}$. Therefore,

$$K \subseteq \{1, 2, \ldots, np_{\max}\}$$

and hence the size of K is linear with respect to n. Let us extend the module described at Fig. 6 in [6] as follows. Let σ_{out} be the output neuron of this module. Let a spike emitted from σ_{out} after halting of the system Π feed two new neurons, each with a rule $a \to a; 0$. Finally, add a new neuron $\sigma_{out'}$ with incoming synapses from these two neurons, another synapse from the original output neuron of Π, and with a rule $a^2 \to a; 0$. Let $\sigma_{out'}$ be the output neuron of Π'. Note that $\sigma_{out'}$ spikes if and only if Π halts and its output neuron σ_{out} does not spike which concludes the proof.

Note that the above proof holds also for a certain subclass of extended SN P systems with $p_{\max}$ bounded from above by $poly(n)$. It is an *open problem* whether an analogous result holds for unrestricted extended SN P systems.

Corollary 1. *The classes* $\mathbf{PSN}_{\mathcal{R}}$ *and* $\mathbf{PSN}^*_{\mathcal{R}}$ *are closed under the operation of complement, for* $\mathcal{R}$ *omitted or* $\mathcal{R} \in \{-reg, -del, ssnf\}$.

Theorem 2. *Let* $\mathcal{R}$ *be an arbitrary class of SN P systems. Let* X *and* Y *be decision problems such that* X *is reducible to* Y *in polynomial time. If* $Y \in \mathbf{PSN}_{\mathcal{R}}$ *(respectively,* $Y \in \mathbf{PSN}^*_{\mathcal{R}}$*), then* $X \in \mathbf{PSN}_{\mathcal{R}}$ *(resp.,* $X \in \mathbf{PSN}^*_{\mathcal{R}}$*).*

Proof. We prove the case of SN P systems with input, adopting the technique used in [14], the case without input is analogous. Let $\mathbf{\Pi}$ by a family providing uniform solution to the problem Y. By its definition, let p be a polynomial and (cod, s) a polynomial encoding of Y in $\mathbf{\Pi}$ such that $\mathbf{\Pi}$ is p-bounded with respect to Y and sound and complete with respect to (Y, cod, s).

Let $r : I_X \to I_Y$ be a polynomial time reduction from X to Y, hence there is a polynomial q such that for each $w \in I_X$, $|r(w)| \leq q(|w|)$. Observe that:

- By Lemma 4, $(cod \circ r; s \circ r)$ is a polynomial encoding of X in $\mathbf{\Pi}$.
- $\mathbf{\Pi}$ is $(p \circ q)$-bounded with respect to X since for each $w \in I_X$, every computation of $\Pi(s(r(w)))$ with input $cod(r(w))$ performs at most $p(|r(w)|) \leq p(q(|w|))$ steps.
- $\mathbf{\Pi}$ is sound and complete with respect to $(X, cod \circ r, s \circ r)$ since for each $w \in I_X$,
  - if there exists an accepting computation of $\Pi(s(r(w)))$ with input $cod(r(w))$, then $\theta_Y(r(w)) = 1$ and, by reduction, also $\theta_X(w) = 1$,
  - if $\theta_X(w) = 1$, then also $\theta_Y(r(w)) = 1$ and hence every computation of $\Pi(s(r(w)))$ with input $cod(r(w))$ is an accepting computation.

Consequently, $X \in \mathbf{SN}_{\mathcal{R}}(p \circ q)$ and hence also in $\mathbf{PSN}_{\mathcal{R}}$.

5 Efficiency of Basic Classes of SN P Systems

As we have already mentioned, no standard SN P system can simulate Turing machine with less than exponential time and space overheads [9]. Therefore, we focus on families of SN P systems in this section. We start with a simple variant of SN P systems with restrictions imposed on their regular expressions. Results in [7] together with Theorems 1 and 4 in [15] imply the following statement.

Theorem 3. $\mathbf{PSN}_{-reg,-del} = \mathbf{PSN}^*_{-reg,-del} = \mathbf{PSN}_{ssnf} = \mathbf{PSN}^*_{ssnf} = \mathbf{P}$

These results show that families of standard confluent SN P systems can reach the computational power beyond $\mathbf{P}$ only with the aid of complex regular expressions. Whenever we release the condition of single star normal forms in regular expressions, the computational power of SN P systems reaches the class $\mathbf{NP}$.

Theorem 4. $(\mathbf{NP} \cup \mathbf{co\text{-}NP}) \subseteq \mathbf{SN}^*_{-del}(2)$

Proof. A part of the statement concerning $\mathbf{NP}$ follows by Proposition 1 in [7] which presents a construction of a standard deterministic SN P system solving the problem Subset Sum in one step. By Theorem 1, also the complement of this problem (which is co-NP-complete) can be solved in the same way. Actually, in this case it is enough to add two more neurons which add one more step of computation.

Corollary 2. $(\mathbf{NP} \cup \mathbf{co\text{-}NP}) \subseteq \mathbf{PSN}^*_{-del}$

Note that Theorem 4 and Corollary 2 hold only for succinct (binary) representation of unary strings in regular expressions and also initial number of spikes in neurons.

Indeed, if one assumes unary representation of regular expressions and of number of spikes in neurons, then uniform families of standard confluent SN P system cannot solve NP-complete problems unless $\mathbf{P} = \mathbf{NP}$. Recall that any confluent SN P system with simple regular expressions can be simulated by a deterministic Turing machine in polynomial time [7]. With the unary representation, one can extend the result also to general regular expressions: an expression

E can be transformed into the equivalent NFA in polynomial time. Then it is decidable in polynomial time with respect to the size of E and k, whether the NFA accepts the string a^k representing k spikes in a neuron.

It remains an *open problem* whether a result analogous to Corollary 2 holds for standard confluent SN P systems with input. We conjecture that this can be achieved only with the aid of maximal parallelism or extended rules which would allow to transform rapidly a binary input to an exponential number of spikes present in some neuron as in [8].

To establish an upper bound on the power of standard confluent families of SN P systems with unlimited regular expressions, we need the following lemma first:

Lemma 5. *Matching of a regular expression E of size s in succinct form over a singleton alphabet with a string a^k can be done on a RAM in non-deterministic time $\mathcal{O}(s \log k)$.*

Proof. Assume that we have the syntactic tree of the expression E at our disposal (its parsing can be done in deterministic polynomial time). We treat the subexpressions of the form a^n as constants and assign them a leaf node of the tree with the value n. The matching algorithm works as follows:

- Produce non-deterministically a random element of $L(E)$ in succinct form by the depth-first search traversal of its syntactic tree: start in the root and evaluate recursively each node:
  - leaf node containing a constant: return the value of the node;
  - catenation: evaluate both subtrees of this node and add the results;
  - union: choose non-deterministically one of the subtrees of this node and evaluate it;
  - star: draw a random number of iterations x within the range $\langle 0, k \rangle$, evaluate the subtree starting in this node and multiply the result by x.
- Compare the drawn element of $L(E)$ with a^k whether they are equal.

Whenever during the evaluation the computed value exceeds k, the algorithm halts immediately and reports that a^k does not match $L(E)$. This guarantees that the number of bits processed in each operation is always $O(\log k)$.

Each of the elementary operations described above can be performed in constant time on RAM with unit instruction cost, except the multiplication which requires $O(\log k)$ time. Total number of tree-traversal steps is $\mathcal{O}(s)$.

Theorem 5. PSN* $\subseteq$ PSPACE

Proof. It has been shown in [7] that any confluent SN P system with simple regular expressions can be simulated by a deterministic Turing machine in polynomial time. Assuming general regular expressions, by Lemma 5 their matching can be done in non-deterministic polynomial time, and since **NP** $\subseteq$ **PSPACE**, also in deterministic polynomial space. Indeed, if one replaces the random selection in the proof of Lemma 5 by the depth-first search of all configurations reachable by making nondeterministic choices, one gets a deterministic algorithm running in polynomial space and exponential time.

Denote by s the size of description of an SN P system Π. Observe that the total number of bits to describe spikes in all neurons after t steps of computation is $\mathcal{O}(s+t)$ even in the case of maximal parallelism or exhaustive rules. The total size of all regular expressions in Π is $\mathcal{O}(s)$. Hence, by Lemma 5, the simulation of Π performs in polynomial space with respect to $s+t$.

Finally, let us note that a deterministic solution to PSPACE-complete problems QSAT and Q3SAT with families of SN P systems with pre-computed resources (i.e., with exponential amount of neurons) have been shown in [4].

6 Conclusion

We have introduced uniform families of standard confluent SN P systems and studied their computational power under polynomial time restriction. Several factors were focused on, influencing the obtained results: the input encoding, the form of output (halting versus spiking), the descriptional complexity, the form of regular expressions.

It was shown that, with the restriction of regular expressions to the single star normal form, these families of SN P systems characterize the class **P**. It remains an open problem whether this condition can be further relaxed.

When complex regular expressions are allowed (but note that the operation * is not necessary), these families are capable to solve **NP**-complete problems in constant time. The succinct representation of regular expressions and of spikes in neurons is necessary to achieve this computational potential (unless **P**=**NP**). Finally, the power of these families under polynomial time restriction is bounded from above by **PSPACE**.

The results concerning intractable problems were shown for the case of families without input. It is very likely that the same result for the case of families with input is possible only when extended rules and/or maximal parallelism are allowed. However, there is no formal proof known yet.

Acknowledgements

The research was supported by the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2009–14421, by the Comunidad de Madrid (grant No. CCG06-UPM/TIC-0386 to the LIA research group), by the Czech Science Foundation, grant No. 201/09/P075, and by the Silesian University in Opava, grant No. SGS/4/2010.

References

1. Andrei, S., Cavadini, S.V., Chin, W.-N.: A new algorithm for regularizing one-letter context-free grammars. Theoretical Computer Science 306, 113–122 (2003)
2. Gutiérrez-Naranjo, M.A., Leporati, A.: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. In: Díaz-Pernil, D., et al. (eds.) Proceedings of Sixth Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, pp. 193–210 (2008)

3. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71(2–3), 279–308 (2006)
4. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., Zhang, X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. In: Martínez-del-Amor, M.A., et al. (eds.) Seventh Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, vol. 2, pp. 1–27 (2009)
5. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., Zhang, X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. In: Theoretical Computer Science (in Press, 2010)
6. Leporati, A., Mauri, G., Zandron, C., Păun, G., Pérez-Jiménez, M.J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. Natural Computing 8(4), 681–702 (2009)
7. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: On the computational power of spiking neural P systems. In: Gutiérrez-Naranjo, M.A., Păun, G., Romero-Jiménez, A., Riscos-Núnez, A. (eds.) Proceedings of Fifth Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, pp. 227–245 (2007)
8. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 336–352. Springer, Heidelberg (2007)
9. Neary, T.: On the computational complexity of spiking neural P systems. In: Calude, C.S., Costa, J.F., Freund, R., Oswald, M., Rozenberg, G. (eds.) UC 2008. LNCS, vol. 5204, pp. 189–205. Springer, Heidelberg (2008)
10. The P Systems Web Page, http://ppage.psystems.eu/, [cit. 2009-12-29]
11. Păun, G., Pérez-Jiménez, M.J., Rozenberg, G.: Spike trains in spiking neural P systems. Intern. J. Found. Computer Sci. 17(4), 975–1002 (2006)
12. Ibarra, O.H., Leporati, A., Păun, A., Woodworth, S.: Spiking neural P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 337–362. Oxford University Press, Oxford (2009)
13. Pérez-Jiménez, M.J.: A computational complexity theory in membrane computing. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 125–148. Springer, Heidelberg (2010)
14. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics 11(4), 423–434 (2006)
15. Rodríguez-Patón, A., Sosík, P., Cienciala, L.: On complexity clases of spiking neural P systems. In: Martínez-del-Amor, et al. (eds.) Proceedings of Eighth Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, pp. 267–282 (2010)

Spiking Neural P Systems with Neuron Division

Jun Wang[1,2], Hendrik Jan Hoogeboom[2], and Linqiang Pan[1,⋆]

[1] Image Processing and Intelligent Control Key Laboratory of Education Ministry
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
junwangjf@gmail.com, lqpan@mail.hust.edu.cn

[2] Leiden Institute of Advanced Computer Science, Universiteit Leiden,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
hoogeboom@liacs.nl

Abstract. Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. The features of neuron division and neuron budding are recently introduced into the framework of SN P systems, and it was shown that SN P systems with neuron division and neuron budding can efficiently solve computationally hard problems. In this work, the computation power of SN P systems with neuron division only, without budding, is investigated; it is proved that a uniform family of SN P systems with neuron division can efficiently solve SAT in a deterministic way, not using budding, while additionally limiting the initial size of the system to a constant number of neurons. This answers an open problem formulated by Pan et al.

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [4] as a new class of distributed and parallel computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons (see, e.g., [3, 11, 12]). The resulting models are a variant of tissue-like and neural-like P systems from membrane computing. Please refer to the classic [14] for the basic information about membrane computing, to the handbook [15] for a comprehensive presentation, and to the web site [16] for the up-to-date information.

In short, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph, which send signals (*spikes*) along *synapses* (arcs of the graph). Each neuron contains a number of spikes, and is associated with a number of *firing* and *forgetting rules*: within the system the spikes are moved, created, or deleted.

The computational efficiency of SN P systems has been recently investigated in a series of works [1, 5–10]. An important issue is that of *uniform* solutions to

⋆ Corresponding author. Tel.: +86-27-87556070; Fax: +86-27-87543130.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 361–376, 2010.

NP-complete problems, i.e., where the construction of the system depends on the problem and not directly on the specific problem instance (if may, however, depend on the size of the instance). Within this context, most of the solutions exploit the power of nondeterminism [8–10] or use pre-computed resources of exponential size [1, 5–7].

Recently, another idea is introduced for constructing SN P systems to solve computationally hard problems by using neuron division and budding [13], where for all $n, m \in \mathbb{N}$ all the instances of $SAT(n, m)$ with at most n variables and at most m clauses are solved in a deterministic way in polynomial time using a polynomial number of initial neurons. As both neuron division rules and neuron budding rules are used to solve $SAT(n, m)$ problem in [13], it is a natural question to design efficient SN P systems omitting either neuron division rules or neuron budding rules for solving **NP**-complete problem.

In this work, a uniform family of SN P systems with only neuron division is constructed for efficiently solving `SAT` problem, which answers the above question posed in [13]. Additionally, the result of [13] is improved in the sense that the SN P systems are constructed with a constant number of initial neurons instead of linear number with respect to the parameter n, while the computations still last a polynomial number of steps.

2 SN P Systems with Neuron Division

Readers are assumed to be familiar with basic elements about SN P systems, e.g., from [4] and [16], and formal language theory, as available in many monographs. Here, only SN P systems with only neuron division are introduced.

A *spiking neural P system with neuron division* is a construct Π of the following form:

$$\Pi = (\{a\}, H, syn, n_1, \ldots, n_m, R, in, out), \text{ where:}$$

1. $m \geq 1$ (the initial degree of the system);
2. a is an object, called *spike*;
3. H is a finite set of *labels* for neurons;
4. $syn \subseteq H \times H$ is a *synapse dictionary* between neurons; with $(i, i) \notin syn$ for $i \in H$;
5. $n_i \geq 0$ is the *initial number of spikes* contained in neuron i, $i \in \{1, 2, \ldots, m\}$;
6. R is a finite set of *developmental rules*, of the following forms:
   (1) *extended firing* (also *spiking*) rule $[E/a^c \to a^p; d]_i$, where $i \in H$, E is a regular expression over a, and $c \geq 1$, $p \geq 0$, $d \geq 0$, such that $c \geq p$;
   (2) *neuron division rule* $[E]_i \to [\ ]_j \parallel [\ ]_k$, where E is a regular expression over a and $i, j, k \in H$;
7. $in, out \in H$ indicate the *input* and the *output* neurons of Π.

Several shorthand notations are customary for SN P systems. If a rule $[E/a^c \to a^p; d]_i$ has $E = a^c$, then it is written in the simplified form $[a^c \to a^p; d]_i$; similarly,

if it has $d = 0$, then it is written as $[E/a^c \to a^p]_i$; of course notation for $E = a^c$ and $d = 0$ can be combined into $[a^c \to a^p]_i$. A rule with $p = 0$ is called *extended forgetting* rule.

If a neuron σ_i (a notation used to indicate a neuron that has label i) contains k spikes and $a^k \in L(E), k \geq c$, where $L(E)$ denotes the language associated with the regular expression E, then the rule $[E/a^c \to a^p; d]_i$ is enabled and it can be applied. It means that c spikes are consumed, $k - c$ spikes remain in the neuron, and p spikes are produced after d time units. If $d = 0$, then the spikes are emitted immediately; if $d \geq 1$ and the rule is used in step t, then in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is closed and it cannot receive new spikes (these particular input spikes are "lost"; that is, they are removed from the system). In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes. Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that there is a synapse going from σ_i to σ_j, i.e., $(\sigma_i, \sigma_j) \in syn$, and which are open. Of course, if neuron σ_i has no synapse leaving from it, then the produced spikes are lost. If the rule is a forgetting one of the form $[E/a^c \to \lambda]_i$, then, when it is applied, $c \geq 1$ spikes are removed, but none are emitted.

If a neuron σ_i contains s spikes and $a^s \in L(E)$, then the division rule $[E]_i \to [\,]_j \parallel [\,]_k$ can be applied, consuming s spikes the neuron σ_i is divided into two neurons, σ_j and σ_k. The child neurons contain no spike in the moment when they are created, but they contain developmental rules from R and inherit the synapses that the parent neuron already has; that is, if there is a synapse from neuron σ_g to the parent neuron σ_i, then in the process of division, one synapse from neuron σ_g to child neuron σ_j and another one from σ_g to σ_k are established. The same holds when the connections are in the other direction (from the parent neuron σ_i to a neuron σ_g). In addition to the inheritance of synapses, the child neurons can have new synapses as provided by the synapse dictionary. If a child neuron $\sigma_g, g \in \{j, k\}$, and another neuron σ_h have the relation $(g, h) \in syn$ or $(h, g) \in syn$, then a synapse is established between neurons σ_g and σ_h going from or coming to σ_g, respectively.

In each time unit, if a neuron can use one of its rules, then a rule from R must be used. For the general model, if several rules are enabled in the same neuron, then only one of them is chosen non-deterministically. In this paper however all neurons behave deterministically and there will be no conflict between rules. When a neuron division rule is applied, at this step the associated neuron is closed, it cannot receive spikes. In the next step, the neurons obtained by division will be open and can receive spikes. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel.

The *configuration* of the system is described by the topological structure of the system, the number of spikes associated with each neuron, and the *state* of each neuron (open or closed). Using the rules as described above, one can define *transitions* among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation *halts* if it reaches a configuration where all neurons are open and no rule can be used.

If m is the initial degree of the system, then the initial configuration of the system consists of neurons $\sigma_1, \ldots, \sigma_m$ with labels $1, \ldots, m$ and connections as specified by the synapse dictionary *syn* for these labels. Initially, $\sigma_1, \ldots, \sigma_m$ contain $n_1, \ldots, n_m$ spikes, respectively.

In the next section, the input of a system is provided by a sequence of spikes entering the system in a number of consecutive steps via the input neuron. Such a sequence is written in the form $a^{i_1}.a^{i_2}.\cdots.a^{i_r}$, where $r \geq 1, i_j \geq 0$ for each $1 \leq j \leq r$, which means that i_j spikes are introduced in neuron σ_{in} in step j of a computation.

3 Solving SAT

In this section, a uniform family of SN P systems with neuron division is constructed for efficiently solving SAT, the most invoked **NP**-complete problem [2]. The instances of SAT consist of two parameters: the number n of variables and a propositional formula which is a conjunction of m clauses, $\gamma = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Each *clause* is a disjunction of literals, occurrences of x_i or $\neg x_i$, built on the set $X = \{x_1, x_2, \ldots, x_n\}$ of variables. An *assignment* of the variables is a mapping $p : X \to \{0, 1\}$ that associates to each variable a truth value. We say that an assignment p *satisfies* the formula γ if, once the truth values are assigned to all the variables according to p, the evaluation of γ gives 1 (*true*) as a result (meaning that in each clause at least one of the literals must be true).

The set of all instances of SAT with n variables and m clauses is denoted by $SAT(n, m)$. Because the construction is uniform, any given instance γ of $SAT(n, m)$ needs to be encoded. Here, the way of encoding given in [13] is followed. As each clause C_i of γ is a disjunction of at most n literals, for each $j \in \{1, 2, \ldots, n\}$ either x_j occurs in C_i, or $\neg x_j$ occurs, or none of them occurs. In order to distinguish these three situations the *spike variables* $\alpha_{i,j}$ are defined, for $1 \leq i \leq m$ and $1 \leq j \leq n$, whose values are the amounts of spikes assigned as follows:

$$\alpha_{i,j} = \begin{cases} a & \text{if } x_j \text{ occurs in } C_i, \\ a^2 & \text{if } \neg x_j \text{ occurs in } C_i, \\ a^0 & \text{otherwise.} \end{cases}$$

In this way, clause C_i will be represented by the sequence $\alpha_{i,1}.\alpha_{i,2}.\ldots.\alpha_{i,n}$ of spike variables. In order to give the systems enough time to generate the necessary workspace before computing the instances of $SAT(n, m)$, a spiking train $(a^0.)^{4n}$ is added in front of the formula encoding spike train. Thus, for any given instance γ of $SAT(n, m)$, the encoding sequence equals $cod(\gamma) = (a^0.)^{4n}\ \alpha_{1,1}.\alpha_{1,2}.\ldots.\alpha_{1,n}.\alpha_{2,1}.\alpha_{2,2}.\ldots.\alpha_{2,n}.\ldots.\alpha_{m,1}.\alpha_{m,2}.\ldots.\alpha_{m,n}$.

For each $n, m \in \mathbb{N}$, a system of initial degree 11 is constructed,

$$\Pi(\langle n, m \rangle) = (\{a\}, H, syn, n_1, \ldots, n_{11}, R, in, out),$$

with the following components:

$H = \{in, out\} \cup \{0, 1, 2, 3, 4\}$
$\cup \{b_i \mid i = 1, 2, \ldots, n-1\} \cup \{d_i \mid i = 0, 1, \ldots, n\}$
$\cup \{e_i \mid i = 1, 2, \ldots, n-1\} \cup \{Cx_i \mid i = 1, 2, \ldots, n\}$
$\cup \{g_i \mid i = 1, 2, \ldots, n-1\} \cup \{h_i \mid i = 1, 2, \ldots, n\}$
$\cup \{Cx_i0 \mid i = 1, 2, \ldots, n\} \cup \{Cx_i1 \mid i = 1, 2, \ldots, n\}$
$\cup \{t_i \mid i = 1, 2, \ldots, n+1\} \cup \{f_i \mid i = 1, 2, \ldots, n+1\}$;
$syn = \{(1, b_1), (1, e_1), (1, g_1), (1, 2), (3, 4), (4, 0), (0, out)\}$
$\cup \{(i+1, i) \mid i = 0, 1, 2\} \cup \{(d_n, d_1), (d_n, 4)\}$
$\cup \{(d_i, d_{i+1}) \mid i = 0, 1, \ldots, n-1\} \cup \{(in, Cx_i) \mid i = 1, 2, \ldots, n\}$
$\cup \{(d_i, Cx_i) \mid i = 1, 2, \ldots, n\} \cup \{(Cx_i, h_i) \mid i = 1, 2, \ldots, n\}$
$\cup \{(Cx_i1, t_i) \mid i = 1, 2, \ldots, n\} \cup \{(Cx_i0, f_i) \mid i = 1, 2, \ldots, n\}$;
the labels of the initial neurons are: $in, out, d_0, b_1, e_1, g_1, 0, 1, 2, 3, 4$; the initial contents are $n_{d_0} = 5$, $n_{b_1} = n_{e_1} = n_{g_1} = n_2 = 2$, $n_3 = 7$, and there is no spike in the other neurons;

R is the following set of rules:

(A) rules for the 'Generation stage':
$[a^2]_{b_i} \to [\,]_{d_i} \parallel [\,]_{b_{i+1}}$, $i = 1, 2, \ldots, n-1$,
$[a^2]_{b_{n-1}} \to [\,]_{d_{n-1}} \parallel [\,]_{d_n}$,
$[a^2]_{e_i} \to [\,]_{Cx_i} \parallel [\,]_{e_{i+1}}$, $i = 1, 2, \ldots, n-1$,
$[a^2]_{e_{n-1}} \to [\,]_{Cx_{n-1}} \parallel [\,]_{Cx_n}$,
$[a^2]_{g_i} \to [\,]_{h_i} \parallel [\,]_{g_{i+1}}$, $i = 1, 2, \ldots, n-1$,
$[a^2]_{g_{n-1}} \to [\,]_{h_{n-1}} \parallel [\,]_{h_n}$,
$[a^2]_{h_i} \to [\,]_{Cx_i1} \parallel [\,]_{Cx_i0}$, $i = 1, 2, \ldots, n$,
$[a \to \lambda]_{d_i}$, $i = 1, 2, \ldots, n$,
$[a^2 \to \lambda]_{d_i}$, $i = 1, 2, \ldots, n$,
$[a \to \lambda]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^2 \to \lambda]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a \to \lambda]_{Cx_i1}$, $i = 1, 2, \ldots, n$,
$[a^2 \to \lambda]_{Cx_i1}$, $i = 1, 2, \ldots, n$,
$[a \to \lambda]_{Cx_i0}$, $i = 1, 2, \ldots, n$,
$[a^2 \to \lambda]_{Cx_i0}$, $i = 1, 2, \ldots, n$,
$[a \to a]_i$, $i = 1, 2$,
$[a^2 \to a^2]_i$, $i = 1, 2$,
$[a^3 \to \lambda]_2$,
$[a^4 \to a]_2$,
$[a^7/a^2 \to a^2; 2n-3]_3$,
$[a^5/a^2 \to a^2; 2n-1]_3$,
$[a^2 \to \lambda]_4$,
$[a^2 \to \lambda]_0$,
$[a]_0 \to [\,]_{t_1} \parallel [\,]_{f_1}$,
$[a]_{t_i} \to [\,]_{t_{i+1}} \parallel [\,]_{f_{i+1}}$, $i = 1, 2, \ldots, n-1$,
$[a]_{f_i} \to [\,]_{t_{i+1}} \parallel [\,]_{f_{i+1}}$, $i = 1, 2, \ldots, n-1$;

(B) rules for the 'Input stage':
$[a \to a]_{in}$,
$[a^2 \to a^2]_{in}$,

$[a^4/a^3 \to a^3; 4n]_{d_0}$,
$[a \to a; nm-1]_{d_0}$,
$[a^3 \to a^3]_{d_i}$, $i = 1, 2, \ldots, n$,
$[a^4 \to \lambda]_{d_1}$,
$[a^3 \to \lambda]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^4 \to a^4; n-i]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^5 \to a^5; n-i]_{Cx_i}$, $i = 1, 2, \ldots, n$;

(C) rules for the 'Satisfiability checking stage':
$[a^4 \to a^3]_{Cx_i 1}$, $i = 1, 2, \ldots, n$,
$[a^5 \to \lambda]_{Cx_i 1}$, $i = 1, 2, \ldots, n$,
$[a^4 \to \lambda]_{Cx_i 0}$, $i = 1, 2, \ldots, n$,
$[a^5 \to a^3]_{Cx_i 0}$, $i = 1, 2, \ldots, n$,
$[a^3 \to a^3; nm+2]_3$,
$[a^3 \to a; 1]_4$,
$[a^6 \to a^2; 1]_4$,
$[a]_{t_n} \to [\,]_{t_{n+1}} \parallel [\,]_{f_{n+1}}$,
$[a^{3k+1} \to \lambda]_{t_i}$, $1 \le k \le n$, $i = 1, 2, \ldots, n$,
$[a^{3k+2}/a^2 \to a^2]_{t_i}$, $1 \le k \le n$, $i = 1, 2, \ldots, n$,
$[a]_{f_n} \to [\,]_{t_{n+1}} \parallel [\,]_{f_{n+1}}$,
$[a^{3k+1} \to \lambda]_{f_i}$, $1 \le k \le n$, $i = 1, 2, \ldots, n$,
$[a^{3k+2}/a^2 \to a^2]_{f_i}$, $1 \le k \le n$, $i = 1, 2, \ldots, n$,
$[a^{3k+1} \to \lambda]_{t_{n+1}}$, $0 \le k \le n$,
$[a^{3k+2} \to \lambda]_{t_{n+1}}$, $0 \le k \le n$,
$[a^{3k+1} \to \lambda]_{f_{n+1}}$, $0 \le k \le n$,
$[a^{3k+2} \to \lambda]_{f_{n+1}}$, $0 \le k \le n$;

(D) rules for the 'Output stage':
$[(a^2)^+/a \to a]_{out}$.

To solve the SAT problem in the framework of SN P systems with neuron division, the strategy consists of four phases, as in [13]: *Generation Stage*, *Input Stage*, *Satisfiability Checking Stage* and *Output Stage*. In the first stage, the neuron division is applied to generate necessary neurons to constitute the input and satisfiability checking modules, i.e., each possible assignment of variables $x_1, x_2, \ldots, x_n$ is represented by a neuron (with associated connections with other neurons by synapses). In the input stage, the system reads the encoding of the given instance of SAT. In the satisfiability checking stage, the system checks whether or not there exists an assignment of variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in the propositional formula C. In the last stage, the system sends a spike to the environment only if the answer is positive; no spikes are emitted in case of a negative answer.

The initial structure of the original system from [13] is shown in Figure 1, where the initial number of neurons is $4n + 7$ and an exponential number of neurons are generated by the neuron division and budding rules. In this work, the initial number of neurons is reduced to constant 11 and only the neuron division rule is used to generate the SN P systems. The division and budding rules are not indicated in Figure 1: the process starts with neuron σ_0 (to the

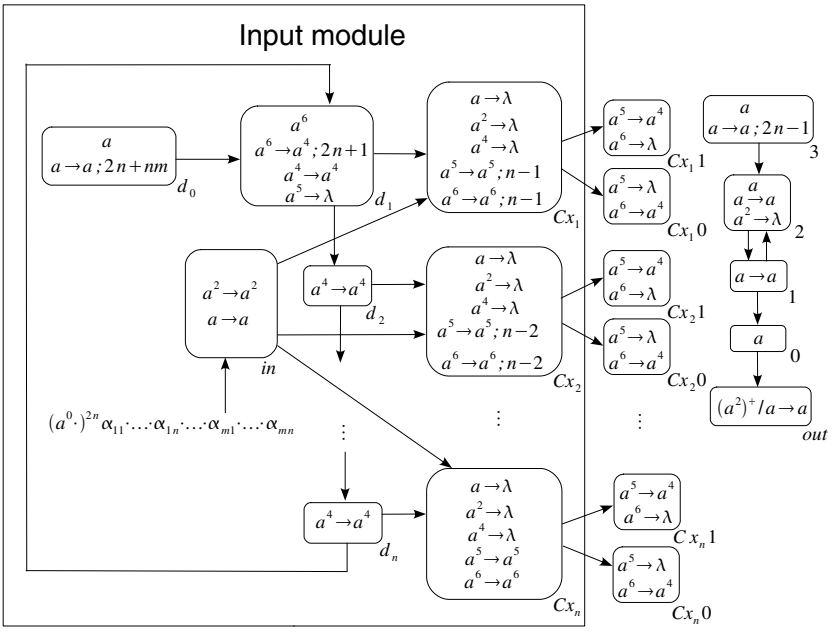


Fig. 1. The initial structure the SN P system Π_2 from [13]

right, before the output neuron), which then results in an exponential number of neurons (with a linear number of fresh labels).

Let us have an overview of the computation. In the initial structure of the system which is shown in Figure 2, there are 11 neurons: the left two neurons σ_{d_0} and σ_{in} are the first layer of the input module; the two neurons σ_{b_1} and σ_{e_1} and their offspring will be used to generate the second and third layers by neuron division rules respectively; the neuron σ_{g_1} and its offspring will be used to generate the first layer of the satisfiability checking module, while neuron σ_0 and its offspring will be used to produce an exponential workspace (the second layer of satisfiability checking module); the auxiliary neurons σ_1, σ_2 and σ_3 supply necessary spikes to the neurons σ_{b_1}, σ_{e_1}, σ_{g_1} and σ_0 and their offspring for neuron division rules; neuron σ_4 supplies spikes to the exponential workspace in the satisfiability checking process; the neuron σ_{out} is used to output the computation result.

By the encoding of instances, it is easy to see that neuron σ_{in} takes $4n$ steps to read $(a^0.)^{4n}$ of $cod(\gamma)$, then the spike variables $\alpha_{i,j}$ will be introduced into the neuron from step $4n+1$. In the first $2n-1$ steps, the system generates the second and third layers of the input module, and also the first layer of the satisfiability checking module; then in the next $2n+1$ steps, neurons σ_0 and its offspring will be used to generate the second layer of satisfiability checking module. After that, the system reads the encoding of the spike variables $\alpha_{i,j}$, checks the satisfiability and outputs the result.

Generation Stage: Neuron σ_{d_0} initially contains 5 spikes and the rule $[a^5/a^4 \to a^4; 4n]_{d_0}$ is applied. It will emit 4 spikes at step $4n+1$ because of the delay $4n$. In

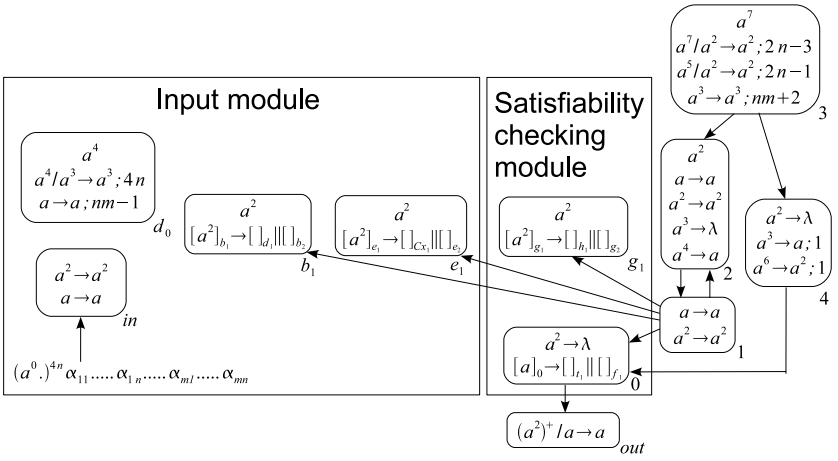


Fig. 2. The initial structure of system $\Pi(\langle n, m\rangle)$

the beginning, neurons σ_{b_1}, σ_{e_1} and σ_{g_1} have 2 spikes respectively, their division rules are applied, and six neurons σ_{d_1}, σ_{b_2}, σ_{Cx_1}, σ_{e_2}, σ_{h_1} and σ_{g_2} are generated. They have six associated synapses $(1, d_1)$, $(1, b_2)$, $(1, Cx_1)$, $(1, e_2)$, $(1, h_1)$ and $(1, g_2)$, where they are obtained by the heritage of synapses $(1, b_1)$, $(1, e_1)$ and $(1, g_1)$, respectively, and three new synapses (d_0, d_1), (d_1, Cx_1) and (Cx_1, h_1) are generated by the synapse dictionary. At step 1, the auxiliary neuron σ_2 sends 2 spikes to neuron σ_1, then in the next step σ_1 will send 2 spikes to neurons σ_{b_2}, σ_{e_2}, σ_{h_1} and σ_{g_2} for the next division. Note that neuron σ_3 has 7 spikes in the beginning, and will send 2 spikes to neuron σ_2 at step $2n-2$ because of the delay $2n-3$ (as we will see, at step $2n-2$, neuron σ_2 also receive 2 spikes from neuron σ_1 and the rule $[a^4 \to a]_2$ will be applied. Hence, after step $2n-1$, neurons σ_0 and its offspring will be used to generate an exponential workspace). The structure of the system after step 1 is shown in Figure 3.

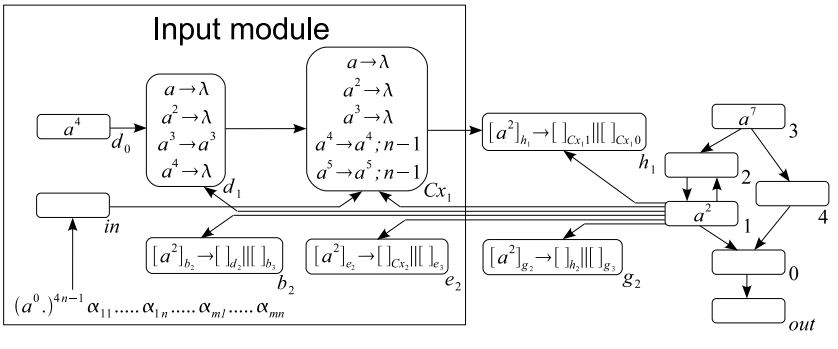


Fig. 3. The structure of system $\Pi(\langle n, m\rangle)$ after step 1

At step 2, neuron σ_1 sends 2 spikes to neurons σ_{b_2}, σ_{e_2}, σ_{h_1}, σ_{g_2}, σ_2, σ_0, σ_{d_1} and σ_{Cx_1}, respectively. In the next step, the former four neurons consume their spikes for neuron division rules; neuron σ_2 sends 2 spikes back to σ_1 (in this way, the auxiliary neurons σ_1, σ_2, σ_3 supply 2 spikes for division for every two steps in the first $2n-2$ steps); the spikes in the other three neurons σ_0, σ_{d_1} and σ_{Cx_1} are deleted by the rules $[a^2 \to \lambda]_{d_1}$, $[a^2 \to \lambda]_{Cx_1}$ and $[a^2 \to \lambda]_0$, respectively. At step 3, neurons σ_{b_2}, σ_{e_2}, σ_{h_1} and σ_{g_2} are divided, eight new neurons σ_{d_2}, σ_{b_3}, σ_{Cx_2}, σ_{e_3}, $\sigma_{Cx_1 1}$, $\sigma_{Cx_1 0}$, σ_{h_2} and σ_{g_3} are generated, and the associated synapses are obtained by heritage or synapse dictionary. The corresponding structure of the system after step 3 is shown in Figure 4.

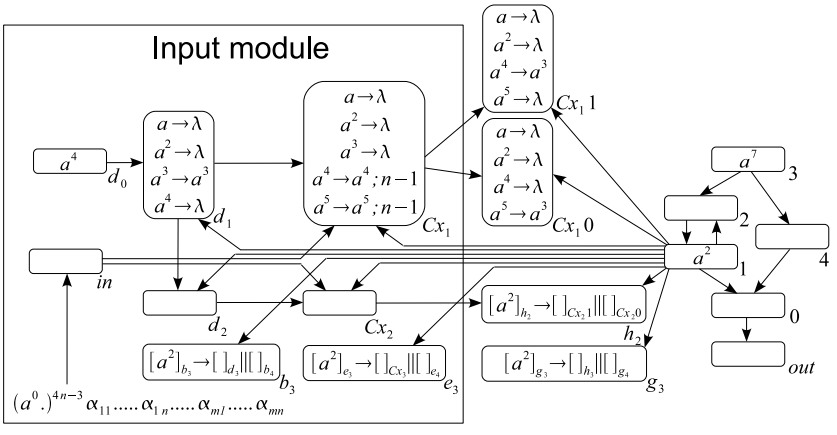


Fig. 4. The structure of system $\Pi(\langle n, m\rangle)$ after step 3

The neuron division is iterated until neurons σ_{d_i}, σ_{Cx_i}, $\sigma_{Cx_i 1}$ and $\sigma_{Cx_i 0}$ $(1 \leq i \leq n)$ are obtained at step $2n-1$. Note that the division rules in neurons $\sigma_{b_{n-1}}$, $\sigma_{e_{n-1}}$ and $\sigma_{g_{n-1}}$ are slightly different with those division rules in neurons σ_{b_i}, σ_{e_i} and σ_{g_i} $(1 \leq i \leq n-2)$. At step $2n-2$, neuron σ_3 sends 2 spikes to neuron σ_2. At the same time, neuron σ_1 also sends 2 spikes to σ_2. So neuron σ_2 sends one spike to σ_1 at step $2n-1$ by the rule $[a^4 \to a]_2$ is applied. Similarly, the auxiliary neurons σ_1, σ_2, σ_3 supply one spike for every two steps to generate an exponential workspace from step $2n-1$ to $4n$ (neuron σ_0 and its offspring use the spikes for division, while neurons σ_{d_i}, σ_{Cx_i}, $\sigma_{Cx_i 1}$ and $\sigma_{Cx_i 0}$ delete the spikes received). Note that the synapses (d_n, d_1) and $(d_n, 4)$ are established by the synapse dictionary. The structure of the system after step $2n-1$ is shown in Figure 5.

At step $2n$, neuron σ_0 has one spike coming from σ_1, the rule $[a]_0 \to [\,]_{t_1} \parallel [\,]_{f_1}$ is applied, and two neurons σ_{t_1}, σ_{f_1} are generated. They have 8 synapses $(1, t_1)$, $(1, f_1)$, $(4, t_1)$, $(4, f_1)$, (t_1, out), (f_1, out), $(Cx1, t_1)$ and $(Cx1 0, f_1)$, where the first 6 synapses are produced by the heritage of the synapses (1, 0), (4, 0) and (0, *out*), respectively; the left two synapses are established by the synapse dictionary. The structure of the system after step $2n+1$ is shown in Figure 6.

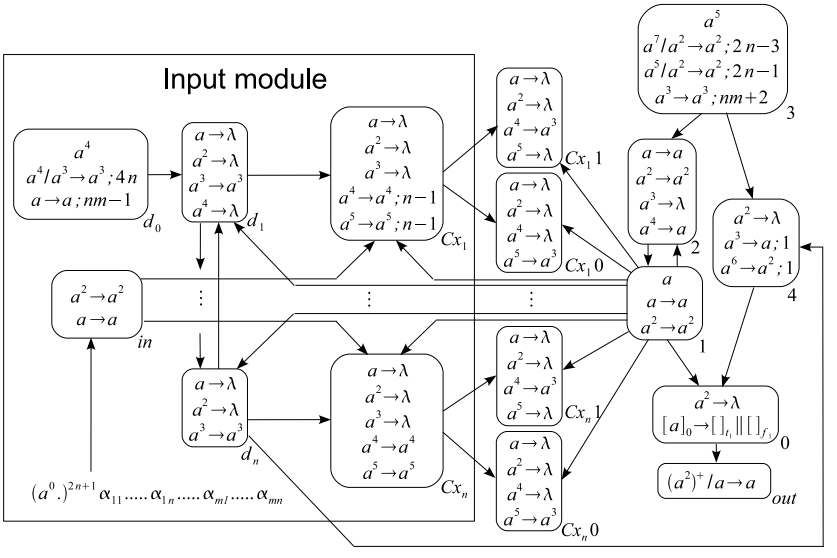


Fig. 5. The structure of system $\Pi(\langle n, m\rangle)$ after step $2n-1$

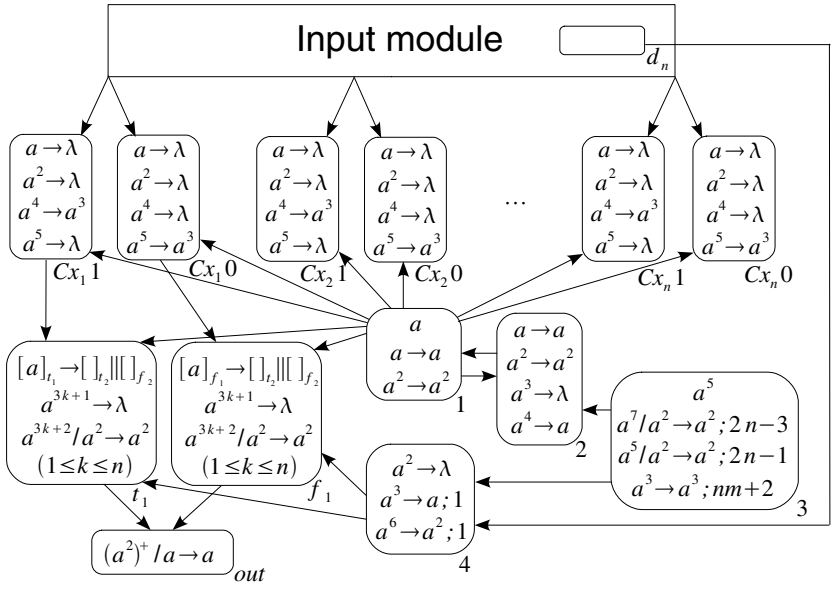


Fig. 6. The structure of system $\Pi(\langle n, m\rangle)$ after step $2n+1$

At step $2n+2$, neurons σ_{t_1} and σ_{f_1} obtain one spike from neuron σ_1 respectively, then in the next step, only division rules can be applied in σ_{t_1} and σ_{f_1}. So these two neurons are divided into four neurons with labels t_2 or f_2 corresponding to assignments $x_1 = 1$ and $x_2 = 1$, $x_1 = 1$ and $x_2 = 0$, $x_1 = 0$ and $x_2 = 1$, $x_1 = 0$ and $x_2 = 0$, respectively. The neuron σ_{Cx_11} (encoding that x_1 appears in a clause) has synapses from it to neurons whose corresponding assignments have $x_1 = 1$. It means that assignments with $x_1 = 1$ satisfy clauses where x_1 appears. The structure of the system after step $2n+3$ is shown in Figure 7.

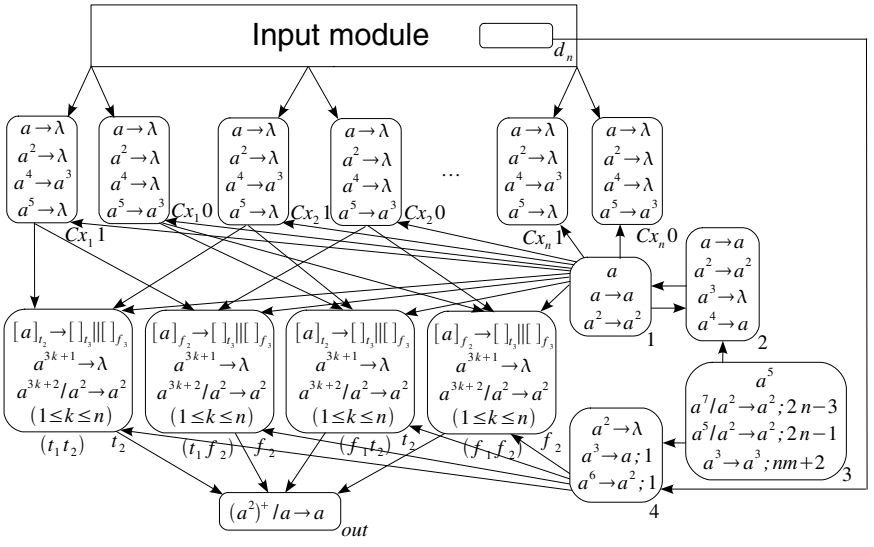


Fig. 7. The structure of system $\Pi(\langle n, m\rangle)$ after step $2n+3$

The exponential workspace is produced by neuron division rules until 2^n neurons with labels t_n or f_n appear at step $4n-1$. At step $4n-2$, neuron σ_3 sends 2 spikes to neurons σ_2 and σ_4 (the rule $[a^5/a^2 \to a^2; 2n-1]_3$ is applied at step $2n-2$), while neuron σ_1 also send one spike to σ_2. So the spikes in neurons σ_2 and σ_4 are deleted by the rules $[a^3 \to \lambda]_2$ and $[a^2 \to \lambda]_4$. The auxiliary neurons σ_1 and σ_2 cannot supply spikes any more and the system passes to read the encoding of a given instance. The structure of the system after step $4n-1$ is shown in Figure 8.

Input Stage: The input module now consists of $2n+2$ neurons, which are in the layers 1 – 3 as illustrated in Figure 5; σ_{in} is the unique input neuron. The spikes of the encoding sequence $code(\gamma)$ are introduced into σ_{in} one "package" by one "package", starting from step 1. It takes $4n$ steps to introduce $(a^0.)^{4n}$ into neuron σ_{in}. At step $4n+1$, the value of the first spike variable α_{11}, which is the virtual symbol that represents the occurrence of the first variable in the first clause, enters into neuron σ_{in}. At the same time, neuron σ_{d_0} sends 3 spikes to neuron σ_{d_1} (the rule $[a^4/a^3 \to a^3; 4n]_{d_0}$ is used at the first step of the computation). At

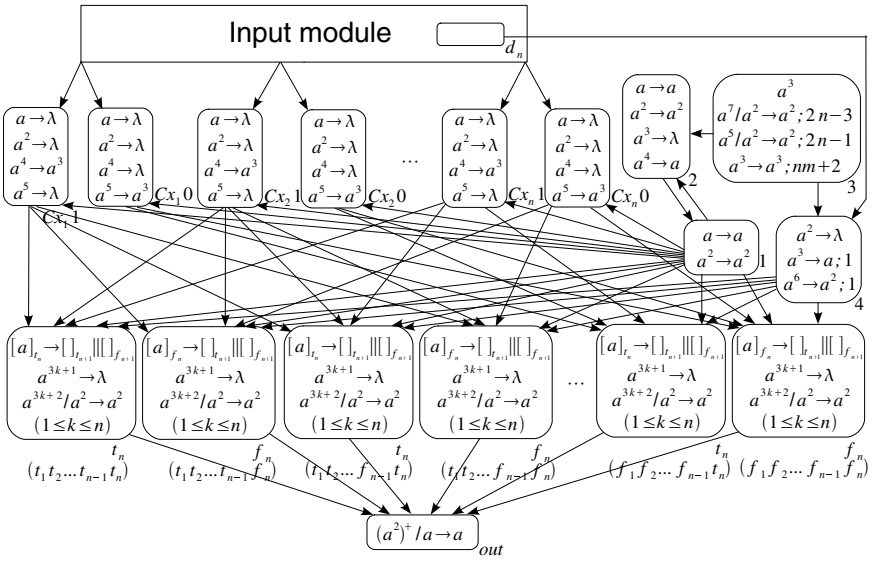


Fig. 8. The structure of system $\Pi(\langle n, m \rangle)$ after step $4n - 1$

step $4n + 2$, the value of the spike variable α_{11} is replicated and sent to neurons σ_{Cx_i}, for all $i \in \{1, 2, \ldots, n\}$, while neuron σ_{d_1} sends 3 auxiliary spikes to neurons σ_{Cx_1} and σ_{d_2}. Hence, neuron σ_{Cx_1} will contain 3, 4 or 5 spikes: if x_1 occurs in C_1, then neuron σ_{Cx_1} collects 4 spikes; if $\neg x_1$ occurs in C_1, then it collects 5 spikes; if neither x_1 nor $\neg x_1$ occur in C_1, then it collects 3 spikes. Moreover, if neuron σ_{Cx_1} has received 4 or 5 spikes, then it will be closed for $n - 1$ steps, according to the delay associated with the rules in it; on the other hand, if 3 spikes are received, then they are deleted and the neuron remains open. At step $4n + 3$, the value of the second spike variable α_{12} from neuron σ_{in} is distributed to neurons σ_{Cx_i}, $2 \le i \le n$, where the spikes corresponding to α_{11} are deleted by the rules $[a \to \lambda]_{Cx_i}$ and $[a^2 \to \lambda]_{Cx_i}$, $2 \le i \le n$. At the same time, the 3 auxiliary spikes are duplicated and one copy of them enters into neurons σ_{Cx_2} and σ_{d_3}, respectively. The neuron σ_{Cx_2} will be closed for $n - 2$ steps only if it contains 4 or 5 spikes, which means that this neuron will not receive any spike during this period. In neurons σ_{Cx_i}, $3 \le i \le n$, the spikes represented by α_{12} are forgotten in the next step.

In this way, the values of the spike variables are introduced and delayed in the corresponding neurons until the value of the spike variable α_{1n} of the first clause and the 3 auxiliary spikes enter together into neuron σ_{Cx_n} at step $5n + 1$ (note that neuron σ_4 also obtains 3 spikes from neuron σ_{d_n} at the same step and will send one spike to the exponential workspace). At that moment, the representation of the first clause of γ has been entirely introduced in the system, and the second clause starts to enter into the input module. In general, it takes $mn + 1$ steps to introduce the whole sequence $code(\gamma)$ in the system, and the input process is completed at step $4n + nm + 1$.

At step $4n + nm + 1$, the neuron σ_{d_n} sends 3 spikes to neuron σ_{d_1}, while the auxiliary neuron σ_{d_0} also sends a spike to the neuron σ_{d_1} (the rule $[a \to a; nm - 1]_{d_0}$ is applied at step $4n + 1$). So neuron σ_{d_1} contains 4 spikes, and in the next step these spikes are forgotten by the rule $[a^4 \to \lambda]_{d_1}$. It ensures that the system eventually halts.

Satisfiability Checking Stage: At step $5n + 1$, all the values of spike variables α_{1i} $(1 \leq i \leq n)$, representing the first clause, have appeared in their corresponding neurons σ_{Cx_i} in the third layer, together with a copy of the 3 auxiliary spikes. In the next step, all the spikes contained in σ_{Cx_i} are duplicated and sent simultaneously to the pair of neurons σ_{Cx_i1} and σ_{Cx_i0} $(1 \leq i \leq n)$ in the first layer of the satisfiability checking module. In this way, each neuron σ_{Cx_i1} and σ_{Cx_i0} receives 4 or 5 spikes when x_i or $\neg x_i$ occurs in C_1, respectively, whereas it receives no spikes when neither x_i nor $\neg x_i$ occurs in C_1.

In general, if neuron σ_{Cx_i1} $(1 \leq i \leq n)$ receives 4 spikes, then the literal x_i occurs in the current clause (say C_j), and thus the clause is satisfied by all those assignments in which x_i is true. Neuron σ_{Cx_i0} will also receive 4 spikes, but they will be deleted during the next computation step. On the other hand, if neuron σ_{Cx_i1} receives 5 spikes, then the literal $\neg x_i$ occurs in C_j, and the clause is satisfied by those assignments in which x_i is false. Since neuron σ_{Cx_i1} is designed to process the case in which x_i occurs in C_j, it will delete its 5 spikes. However, neuron σ_{Cx_i0} has received 5 spikes, and this time it will send 3 spikes to those neurons which are bijectively associated with the assignments for which x_i is false (refer to the generation stage for the corresponding synapses). Note that, neuron σ_4 has 3 spikes at step $5n + 1$, the rule $[a^3 \to a; 1]_4$ is applied, one spike is duplicated and each spike enters into 2^n neurons with labels t_n or f_n at step $5n + 3$ because of the delay 1.

In this way, each neuron with label t_n or f_n receives 1 or $3k + 1$ spikes $(1 \leq k \leq n)$ at step $5n + 3$. If one of neurons σ_{t_n} or σ_{f_n} (we assume the assignment of the neuron is $t_1 t_2 \dots t_{n-1} f_n$) receives 1 spike, which means that none of the neurons σ_{Cx_i1} and σ_{Cx_i0} $(1 \leq i \leq n)$ send spikes to this neuron, then the first clause C_1 is not satisfied by the assignment $t_1 t_2 \dots t_{n-1} f_n$ and this neuron cannot be used to check whether other clauses C_j $(2 \leq j \leq m)$ are satisfied or not. So the neuron with corresponding assignment $t_1 t_2 \dots t_{n-1} f_n$ is divided into two neurons $\sigma_{t_{n+1}}$ and $\sigma_{f_{n+1}}$ by the rule $[a]_{f_n} \to [\,]_{t_{n+1}} \parallel [\,]_{f_{n+1}}$, and the two new neurons cannot send any spike to the output neuron because they will delete the received spikes by the rules $[a^{3k+1} \to \lambda]_{t_{n+1}}$, $[a^{3k+2} \to \lambda]_{t_{n+1}}$, $[a^{3k+1} \to \lambda]_{f_{n+1}}$ and $[a^{3k+2} \to \lambda]_{f_{n+1}}$, with $0 \leq k \leq n$. On the other hand, if a neuron (it is assumed that the assignment of the neuron is $t_1 t_2 \dots t_{n-1} t_n$) receives $3k + 1$ spikes from neurons σ_{Cx_i1} and σ_{Cx_i0}, then these spikes will be forgotten, with the meaning that the clause C_1 is satisfied by the assignment $t_1 t_2 \dots t_{n-1} t_n$ (note that the number of spikes received in neurons with label t_n or f_n is not more than $3n + 1$, because, without loss of generality, we assume that the same literal is not repeated and at most one of literals x_i or $\neg x_i$, for any $1 \leq i \leq n$, can occur in a clause; that is, a clause is a disjunction of at

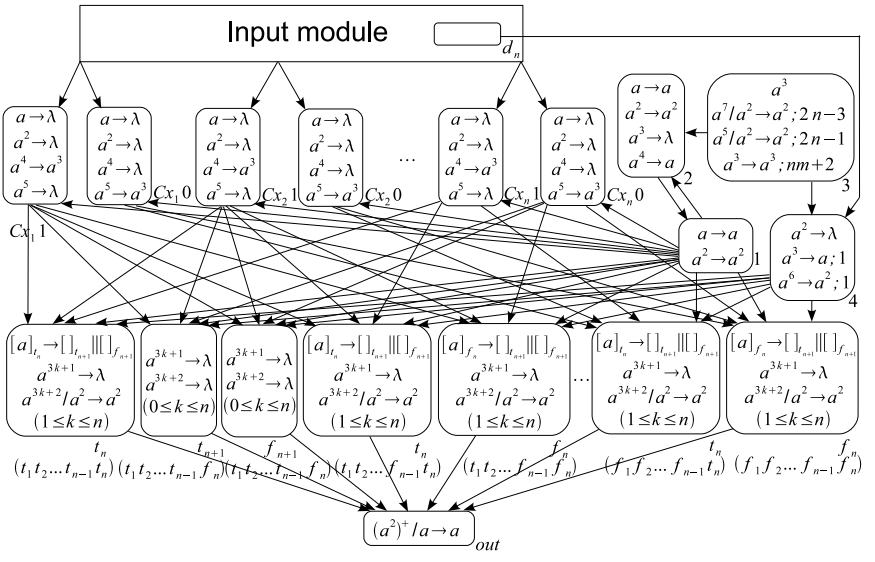


Fig. 9. The structure of system $\Pi(\langle n, m\rangle)$ after step $5n + 4$

most n literals). This occurs in step $(5n + 4)$. In this way, the satisfiability checking for the first clause has been done in $5n + 4$ steps The structure of the system after step $5n + 4$ is shown in Figure 9 (it is assumed only the neuron with corresponding assignment $t_1 t_2 \ldots t_{n-1} f_n$ is divided).

In a similar way, satisfiability checking for the next clause can proceed, and so on. Thus, the first $m - 1$ clauses can be checked to see whether there exist assignments that satisfy all of them. If there exist some assignments that satisfy the first $m - 1$ clauses, which means that their corresponding neurons never receive a spike during the satisfiability checking process of the $m - 1$ clauses.

At step $4n + nm + 1$, the spike variable $\alpha_{m,n}$ of the last clause C_m and the 3 auxiliary spikes (coming from neuron σ_{d_n}) enter into neuron σ_{Cx_n}. At the same moment, neuron σ_4 receives 3 spikes from neuron σ_{d_n} and another 3 spikes from neuron σ_3 (the rule $[a^3 \to a^3; nm + 2]_3$ is applied at step $4n - 2$). So neuron σ_4 contains 6 spikes, and sends 2 spikes to the neurons with labels t_n, f_n, t_{n+1} or f_{n+1} at step $4n + nm + 3$ because of the delay 1. In this way, if neurons with labels t_n and f_n receive $3k + 2$ spikes $(1 \le k \le n)$, with the meaning that these neurons associated with the assignments satisfy all the clauses of γ, then the rules $[a^{3k+2}/a^2 \to a^2]_{t_n}$ and $[a^{3k+2}/a^2 \to a^2]_{f_n}$ can be applied, sending 2 spikes to neuron σ_{out}, respectively. On the other hand, if neurons with labels t_n and f_n receive only 2 spikes, or if neurons with labels t_{n+1} and f_{n+1} receive $3k + 2$ spikes $(0 \le k \le n)$, none of they can send spikes to the output neuron because they cannot satisfy all the clauses of γ. In this way, the satisfiability checking module can complete its process at step $4n + nm + 4$.

Output Stage: From the above explanation of a computation processes, it is not difficult to see that the neuron σ_{out} receives spikes if and only if the formula γ is *true*. At step $4n + nm + 5$, the output neuron sends exactly one spike to the environment if and only if the formula γ is *true*.

According to the above four stages, one can see that the system correctly answers the question whether or not the formula γ is satisfiable. The duration of the computation is polynomial in term of n and m: the system sends one spike to the environment at step $4n + mn + 5$ if the answer is positive; otherwise, the system does not send any spike to the environment and halts in $4n + mn + 5$ steps.

The following is a comparison of the resources used in the systems constructed in this work and in [13]:

Resources \Systems	Systems from this work	Systems from [13]
Initial number of neurons	11	$4n + 7$
Initial number of spikes	20	9
Number of neuron labels	$10n + 7$	$6n + 8$
Size of synapse dictionary	$6n + 11$	$7n + 6$
Number of rules	$2n^2 + 26n + 26$	$n^2 + 14n + 12$

From the above comparison, it is easy to see that the amount of necessary resources for defining each system in this work is polynomial with respect to n. Note that the sets of rules associated with the system $\Pi(\langle n, m\rangle)$ are recursive. Hence, the family $\Pi = \{\Pi(\langle n, m\rangle) \mid n, m \in \mathbb{N}\}$ is polynomially uniform by deterministic Turing machines.

The result of [13] is improved in the sense that a constant number of initial neurons (instead of linear number) are used to construct the systems for efficiently solving `SAT` problem, and the neuron budding rules are not used.

4 Conclusions and Remarks

In this work, a uniform family of SN P systems with only neuron division (not using neuron budding) is constructed for efficiently solving `SAT` problem, which answers an open problem posed in [13]. It is interesting that a constant number of initial neurons are used to construct the systems.

There remain many open problems and research topics about neuron division and neuron budding. As we know, all **NP** problems can be reduced to an **NP**-complete problem in a polynomial time. In principle, if a family of P systems can efficiently solve an **NP**-complete problem, then this family of P systems can also efficiently solve all **NP** problems. But, until now, it remains open how a family of P systems can be designed to efficiently compute the reduction from an **NP** problem to an **NP**-complete problem. Before this open problem is solved, it is still interesting to give efficient solutions to other computationally hard problems in the framework of SN P systems with neuron division.

It is worth investigating the computation power of SN P systems with only neuron budding rules without neuron division rules. Neuron budding can result in a polynomial number of neurons in a polynomial time, and each neuron has a

polynomial number of spikes. It is hard to believe that SN P systems with only neuron budding rules can efficiently solve computationally hard problems unless we happen to find a proof for $\mathbf{P} = \mathbf{NP}$.

Acknowledgements. This work was supported by National Natural Science Foundation of China (61033003 and 30870826), the Fundamental Research Funds for the Central Universities (2010ZD001), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180).

References

1. Chen, H., Ionescu, M., Ishdorj, T.-O.: On the efficiency of spiking neural P systems. In: Proc. 8th Intern. Conf. on Electronics, Information, and Communication, Ulanbator, Mongolia, June 2006, pp. 49–52 (2006)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability. In: A Guide to the Theory of NP-completeness. W.H. Freeman and Company, San Francisco (1979)
3. Gerstner, M., Kistler, W.: Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge University Press, Cambridge (2002)
4. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71(2-3), 279–308 (2006)
5. Ishdorj, T.-O., Leporati, A.: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. Natural Computing 7(4), 519–534 (2008)
6. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., Zhang, X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. Theoretical Computer Science 411(25), 2345–2358 (2010)
7. Leporati, A., Gutiérrez-Naranjo, M.A.: Solving Subset Sum by spiking neural P systems with pre-computed resources. Fundamenta Informaticae 87(1), 61–77 (2008)
8. Leporati, A., Mauri, G., Zandron, C., Păun, G., Pérez-Jiménez, M.J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. Natural Computing 8(4), 681–702 (2009)
9. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 336–352. Springer, Heidelberg (2007)
10. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: On the computational power of spiking neural P systems. International Journal of Unconventional Computing 5(5), 459–473 (2009)
11. Maass, W.: Computing with spikes. Special Issue on Foundations of Information Processing of TELEMATIK 8(1), 32–36 (2002)
12. Maass, W., Bishop, C. (eds.): Pulsed Neural Networks. MIT Press, Cambridge (1999)
13. Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. In: 7th Brainstorming Week on Membrane Computing, vol. II, pp. 151–168 (2009)
14. Păun, G.: Membrane Computing – An Introduction. Springer, Berlin (2002)
15. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press, Cambridge (2010)
16. The P System Web Page, http://ppage.psystems.eu

Matrix Representation of Spiking Neural P Systems

Xiangxiang Zeng[1], Henry Adorna[2], Miguel Ángel Martínez-del-Amor[3], Linqiang Pan[1,⋆], and Mario J. Pérez-Jiménez[3]

[1] Image Processing and Intelligent Control Key Laboratory of Education Ministry
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
xzeng@foxmail.com,
lqpan@mail.hust.edu.cn
[2] Department of Computer Science
(Algorithms and Complexity)
University of the Philippines
Diliman 1101 Quezon City, Philippines
hnadorna@dcs.upd.edu.ph
[3] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{mdelamor,marper}@us.es

Abstract. Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. In this work, a discrete structure representation of SN P systems with extended rules and without delay is proposed. Specifically, matrices are used to represent SN P systems. In order to represent the computations of SN P systems by matrices, configuration vectors are defined to monitor the number of spikes in each neuron at any given configuration; transition net gain vectors are also introduced to quantify the total amount of spikes consumed and produced after the chosen rules are applied. Nondeterminism of the systems is assured by a set of spiking transition vectors that could be used at any given time during the computation. With such matrix representation, it is quite convenient to determine the next configuration from a given configuration, since it involves only multiplication and addition of matrices after deciding the spiking transition vector.

1 Introduction

Membrane computing was initiated by Păun [6] and has developed very rapidly (already in 2003, ISI considered membrane computing as "fast emerging

⋆ Corresponding author. Tel.: +86-27-87556070; Fax: +86-27-87543130.

M. Gheorghe et al. (Eds.): CMC 2010, LNCS 6501, pp. 377–391, 2010.

research area in computer science", see http://esi-topics.com). It aims to abstract computing ideas (data structures, operations with data, computing models, etc.) from the structure and the functioning of single cell and from complexes of cells, such as tissues and organs, including the brain. The obtained models are distributed and parallel computing devices, called *P systems*. For updated information about membrane computing, please refer to [8].

This work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short) [3]. SN P systems were inspired by the neurophysiological behavior of neurons (in brain) sending electrical impulses along axons to other neurons, with the aim of incorporating specific ideas from spiking neurons into membrane computing. Generally speaking, in an SN P system the processing elements are called *neurons* and are placed in the nodes of a directed graph, called the *synapse graph*. The content of each neuron consists of a number of copies of a single object type, namely the *spike*. Each neuron may also contain rules which allow to remove a given number of spikes from it, or send spikes (possibly with a delay) to other neurons. The application of every rule is determined by checking the content of the neuron against a regular set associated with the rule.

Representation of P systems by discrete structures has been one topic in the field of membrane computing. One of the promising discrete structures to represent P systems is matrix. Models with matrices as their representation have been helpful to physical scientists – biologists, chemists, physicists, engineers, statisticians, and economists – solving real world problems. Recently, matrix representation was introduced for represent a restricted form of cell-like P systems without dissolution (where only non-cooperative rules are used) [2]. It was proved that with an algebraic representation P systems can be easily simulated and computed backward (that is, to find all the configurations that produce a given one in one computational step).

In this work, a matrix representation of SN P systems without delay is proposed, where configuration vectors are defined to represent the number of spikes in neurons; spiking vectors are used to denote which rules will be applied; a spiking transition matrix is used to describe the skeleton of a system; the transition net gain vectors are also introduced to quantify the total amount of spikes consumed and produced after the chosen rules are applied. With this algebraic representation, matrix transition can be used to compute the next configuration from a given one.

In addition, we consider another variant of SN P systems, SN P systems with weights (WSN P systems, for short), which were introduced in [9]. In these systems, each neuron has a potential and a given threshold, whose values are real (computable) numbers. Each neuron fires when its potential is equal to its threshold; at that time, part of the potential is consumed and a unit potential is produced (a spike). The unit potential is passed to neighboring neurons multiplied with the weights of synapses. The weights of synapses can also be real (computable) numbers. This variant of SN P system allows real numbers to be computed by the system.

A matrix over $\mathbb{R}_C$ is defined to represent WSN P systems, where vectors are defined to represent the potentials in neurons and the application of rules. In particular, when the potential of a neuron is smaller than its spiking threshold, then this potential vanishes (the potential of the neuron is set to zero). Therefore, a forgetting vector is introduced to denote the potential vanishing from neurons. It is also shown that matrix representation is convenient for deciding the next configuration of the system from a given configuration.

The rest of this paper is organized as follows. In the next section, the definition of SN P systems is introduced. In Section 3 the matrix representation of SN P systems is given, and an example is given to illustrate how to represent the computation of an SN P system by matrix transition. In Section 4 the matrix representation method is extended to WSN P systems. Conclusions and remarks are given in Section 5.

2 Spiking Neural P Systems

In this section, a restricted variant of SN P systems, SN P systems without delay, is introduced.

Definition 1. *An SN P system without delay, of degree $m \geq 1$, is a construct of the form*

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

where:

1. *$O = \{a\}$ is the singleton alphabet (a is called spike);*
2. *$\sigma_1, \ldots, \sigma_m$ are neurons, of the form*

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

   where:
   a) *$n_i \geq 0$ is the initial number of spikes contained in σ_i;*
   b) *R_i is a finite set of rules of the following two forms:*
      (1) *$E/a^c \to a^p$, where E is a regular expression over a, and $c \geq 1$, $p \geq 1$, with the restriction $c \geq p$;*
      (2) *$a^s \to \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \to a^p$ of type (1) from R_i, $a^s \notin L(E)$;*
3. *$syn = \{(i, j) \mid 1 \leq i, j \leq m, i \neq j\}$ (synapses between neurons);*
4. *$in, out \in \{1, 2, \ldots, m\}$ indicate the input and output neurons, respectively.*

The rules of type (1) are spiking (also called firing) rules, which are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a^p \in R_i$ can be applied. This means that consuming (removing) c spikes (thus only $k-c$ spikes remain in σ_i), the neuron is fired, and it produces p spikes; these spikes are transported to all neighbor neurons by outgoing synapses. If a rule $E/a^c \to a^p$ has $E = a^c$, then it is written in the simplified form $a^c \to a^p$.

The rules of type (2) are forgetting rules; they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

In each time unit, if a neuron σ_i can apply one of its rules, then a rule from R_i must be applied. Since two spiking rules, $E_1/a^{c_1} \to a^{p_1}$ and $E_2/a^{c_2} \to a^{p_2}$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules are applicable in a neuron. In that case, only one of them is chosen and applied nondeterministically. However, note that, by definition, if a spiking rule is applicable, then no forgetting rule is applicable, and vice versa.

Thus, the rules are applied in the sequential manner in each neuron, at most one in each step, but neurons function in parallel with each other. It is important to notice that the applicability of a rule is established based on the total number of spikes contained in the neuron.

The configuration of the system is described by the number of spikes present in each neuron. Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be applied. The result of a computation is the number of steps elapsed between the first two spikes sent by the output neuron to the environment during the computation.

3 Matrix Representation of SN P Systems

In this section, a matrix representation of SN P systems is given. Based on this representation, it is shown how the computation of SN P systems can be represented by operations with matrices.

As mentioned in the above section, a configuration of the system is described by the number of spikes present in each neuron. Here, vectors are used to represent configurations.

Definition 2 (Configuration Vectors). *Let Π be an SN P system with m neurons, the vector $C_0 = (n_1, n_2, \ldots, n_m)$ is called the* **initial configuration vector** *of Π, where n_i is the amount of the initial spikes present in neuron σ_i, $i = 1, 2, \ldots, m$ before a computation starts.*

*In a computation, for any $k \in \mathbb{N}$, the vector $C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)})$ is called the k***th configuration vector** *of the system, where $n_i^{(k)}$ is the amount of spikes in neuron σ_i, $i = 1, 2, \ldots, m$ after the kth step of the computation.*

In order to describe which rules are chosen and applied in each configuration, *spiking vectors* are defined.

Definition 3 (Spiking Vectors). *Let Π be an SN P system with m neurons and n rules, and $C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)})$ be the kth configuration vector of Π. Assume a total order $d : 1, \ldots, n$ is given for all the n rules, so the rules can be referred as $r_1, \ldots, r_n$. A* **spiking vector** *$s^{(k)}$ is defined as follows:*

$$s^{(k)} = (r_1^{(k)}, r_2^{(k)}, \ldots, r_n^{(k)}),$$

where:

$$r_i^{(k)} = \begin{cases} 1, \textit{ if the regular expression } E_i \textit{ of rule } r_i \textit{ is satisfied by the} \\ \quad \textit{number of spikes } n_j^{(k)} \textit{ (rule } r_i \textit{ is in neuron } \sigma_j\textit{) and} \\ \quad \textit{rule } r_i \textit{ is chosen and applied;} \\ 0, \textit{ otherwise.} \end{cases}$$

In particular, $s^{(0)} = (r_1^{(0)}, r_2^{(0)}, \ldots, r_n^{(0)})$ *is called the* **initial spiking vector**.

The application of each rule will change the number of spikes in some neurons, for example, when the rule $r_i : E/a^c \to a^p$ is applied in neuron σ_j, it consumes c spikes in σ_j, and emits p spikes; these p spikes are immediately delivered to all the neurons σ_s such that $(j, s) \in syn$. Here, a *spiking transition matrix* is defined to denote the amount of spikes consumed (or received) by each neuron via each rule.

Definition 4 (Spiking Transition Matrix). *Let* Π *be an SN P system with* m *neurons and* n *rules, and* $d : 1, \ldots, n$ *be a total order given for all the* n *rules. The* **spiking transition matrix** *of the system* Π, M_Π, *is defined as follows:*

$$M_\Pi = [a_{ij}]_{n \times m},$$

where:

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

In a spiking transition matrix, the row i is associated with the rule $r_i : E/a^c \to a^p$. Assume that the rule r_i is in neuron σ_j. When the rule r_i is applied, it consumes c spikes in neuron σ_j; neuron σ_s ($s \neq j$ and $(j, s) \in syn$) receives p spikes from neuron σ_j; neuron σ_s ($s \neq j$ and $(j, s) \notin syn$) receives no spike from neuron σ_j. By the definition of spiking transition matrix, the entry in the position (i, j) is a negative number; the other entries in the row i are non-negative numbers. So the following observation holds:

Observation 1: each row of a spiking transition matrix has exactly one negative entry.

In a spiking transition matrix, the column i is associated with neuron σ_i. For an SN P system, without loss of generality, it can be assumed that each neuron has at least one rule inside (if a neuron has no rule inside, it just stores spikes, sending no spikes to other neurons or environment, so it can be deleted without any influence to the computational result of the system). Assume the rules in neuron σ_i are $r_m, r_n, \ldots$. These rules consume spikes of neuron σ_i when they are applied. So the corresponding entries $(m, i), (n, i), \ldots$ in the spiking transition matrix are negative numbers, and the following observation holds:

Observation 2: each column of a spiking transition matrix has at least one negative entry.

In the following, it will be shown that how matrices representing SN P systems can be used to represent the computation of SN P systems by operating with matrices. Before the matrix operations for SN P systems are formally defined, a simple example is provided as follows.

Example 1. Let us consider an SN P system $\Pi = (\{a\}, \sigma_1, \sigma_1, \sigma_3, syn, out)$ that generates the set $\mathbb{N}$ of natural numbers excluding 1, where $\sigma_1 = (2, R_1)$, with $R_1 = \{a^2/a \to a, a^2 \to a\}$; $\sigma_2 = (1, R_2)$, with $R_2 = \{a \to a\}$; $\sigma_3 = (1, R_3)$, with $R_3 = \{a \to a, a^2 \to \lambda\}$; $syn = \{(1,2), (1,3), (2,1), (2,3)\}$; $out = 3$. Π is also represented graphically in Figure 1, which may be easier to understand.

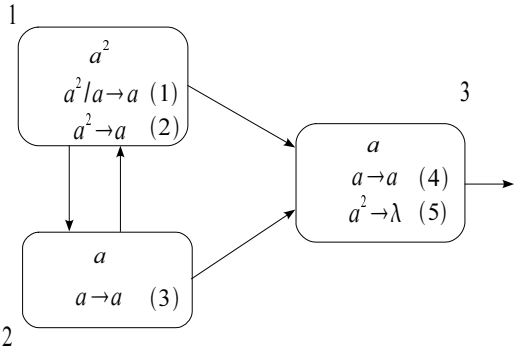


Fig. 1. An SN P system Π that generates the set $\mathbb{N} - \{1\}$

In order to represent the above SN P system Π in a matrix, a total order is set for all the rules in the system, which can be seen in Figure 1. With this order, the rules can be denoted by $r_1, \ldots, r_5$.

Let $M_{\Pi_1} = [a_{ij}]_{5\times 3}$ be the spiking transition matrix for Π. By Definition 4, the row i of M_Π is associated with the rule $r_i : E/a^c \to a^p, c \geq 1, p \geq 0$ in system Π. The entries a_{i1}, a_{i2}, a_{i3} are the amount of spikes which neurons $\sigma_1, \sigma_2, \sigma_3$ will get (or consume) when rule r_i is applied.

The spiking transition matrix for the SN P system Π depicted in Figure 1 is as follows.

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

Initially, neurons σ_1, σ_2, and σ_3 have 2, 1, and 1 spike(s), respectively. According to Definition 2, the initial configuration vector for system Π would be $C_0 = (2, 1, 1)$. Since neuron σ_1 has two rules r_1 and r_2 that are applicable in

the initial transition, one of the rules could be chosen, the initial spiking transition vector would be $(1,0,1,1,0)$ or $(0,1,1,1,0)$ by Definition 3. Note that the rule r_5 is not applicable because the regular expression a^2 is not satisfied in neuron σ_3.

If the rule $r_1 : a^2/a \to a$ is applied, it consumes one spike in neuron σ_1 and sends one spike to neurons σ_2 and σ_3, respectively; at the same time, neuron σ_2 sends one spike to each of the neurons σ_1 and σ_3. In this step, the net gain of neuron σ_1 is 0 spike (it consumes 1 spike by r_1 and receives 1 spike from neuron σ_2); the net gain of neuron σ_2 is 0 spike (it consumes 1 spike by r_3 and receives 1 spike from neuron σ_1); the net gain of neuron σ_3 is 1 spike (it consumes 1 spike by rule r_5 and receives 1 spike from each of the neurons σ_1 and σ_2). After this step, the numbers of spikes in neurons σ_1, σ_2 and σ_3 are 2, 1 and 2, respectively.

The illustration above explained intuitively how an SN P system computes from one configuration to the succeeding one. In order to use matrix operations to represent it, the following definitions are needed:

Definition 5 (Transition Net Gain Vector). *Let Π be an SN P system with m neurons and n rules, and $C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)})$ be the kth configuration vector of Π. The* **transition net gain vector** *at step k is defined as $NG^{(k)} = C_{k+1} - C_k$.*

Lemma 1. *Let Π be an SN P system with m neurons and n rules, $d : 1, \ldots, n$ be a total order for the n rules, M_Π the spiking transition matrix of Π, and $s^{(k)}$ the spiking vector at step k. Then the transition net gain vector at step k can be obtained by*

$$NG^{(k)} = s^{(k)} \cdot M_\Pi. \tag{2}$$

Proof. Assume that $M_\Pi = [a_{ij}]_{m \times n}$, $s^{(k)} = (r_1^{(k)}, r_2^{(k)}, \ldots, r_n^{(k)})$, and $NG^{(k)} = (g_1, g_2, \ldots, g_m)$. Note that the spiking vector $s^{(k)}$ is a $\{0,1\}$-vector that identifies the rules that would be applied at step k. Thus, $\Sigma_{i=1}^{n} r_i^{(k)} a_{ij}$ represents the total amount of spikes received and consumed by neuron σ_j after applying the rules identified by $s^{(k)}$. Therefore, the net gain of neuron σ_j is $g_j = \Sigma_{i=1}^{n} r_i^{(k)} a_{ij}$, for all $j = 1, 2, \ldots, m$. That is, $NG^{(k)} = s^{(k)} \cdot M_\Pi$.

Theorem 1. *Let Π be an SN P system with m neurons and n rules, $d : 1, \ldots, n$ be a total order for the n rules, M_Π the spiking transition matrix of Π, C_k the kth configuration vector, and $s^{(k)}$ the spiking vector at step k, then the configuration C_{k+1} of Π can be obtained by*

$$C_{k+1} = C_k + s^{(k)} \cdot M_\Pi. \tag{3}$$

Proof. This results follows directly from the preceding Lemma.

Let us go back to the example shown in Figure 1. Given the initial configuration vector $C_0 = (2,1,1)$, the next configuration of system Π can be computed as follows.

If the rules r_1, r_3, r_4 are chosen to be applied, the spiking vector is $s^{(0)} = (1,0,1,1,0)$, and the next configuration is

$$C_1 = (2,1,1) \ + \ (1,0,1,1,0) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (2,1,2). \tag{4}$$

In the next step, r_1, r_3, r_5 are chosen to be applied, the spiking vector is $(1,0,1,0,1)$, and the next configuration is

$$C_2 = (2,1,2) \ + \ (1,0,1,0,1) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (2,1,2), \tag{5}$$

where the transition net gain vector is

$$NG = (1,0,1,0,1) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (0,0,0). \tag{6}$$

Equation (6) means that the configuration of the system remains unchanged as long as the rules r_1, r_3 and r_5 are chosen to be applied. However, at any moment, starting from the first step of computation, neuron σ_1 can choose to apply the rule $r_2 : a^2 \to a$. In this case, system will go to another configuration. The checking is left to the readers.

The following Corollary is a direct consequence of the preceding Theorem.

Corollary 1. *Let Π be an SN P system with m neurons and n rules, $d : 1, \ldots, n$ be a total order for the n rules, M_Π the spiking transition matrix of Π, C_k the kth configuration vector, and $s^{(k-1)}$ the spiking vector at step $k-1$, then the previous configuration C_{k-1} is*

$$C_{k-1} = C_k \ - \ s^{(k-1)} \cdot M_\Pi. \tag{7}$$

In the above matrix representation, the spikes sent to the environment are not considered. In order to consider the spikes sent to the environment, an *augmented spiking transition matrix* is introduced.

Definition 6 (Augmented Transition Spiking Matrix). *Let Π be an SN P system with m neurons and n rules, $d : 1, \ldots, n$ be a total order for the n rules, and M_Π the $n \times m$ spiking transition matrix of Π. An* **augmented spiking transition matrix** *is defined as*

$$[M_\Pi \mid e]_{n \times (m+1)},$$

where the column $e = (e_1, e_2, \ldots, e_n)^T$ *represents the spikes sent to the environment, with:*

$$e_i = \begin{cases} p, & \text{if rule } r_i \text{ is in the output neuron and it is applied producing } p \text{ spikes;} \\ 0, & \text{if rule } r_i \text{ is not in the output neuron.} \end{cases}$$

Correspondingly, the **augmented configuration vector** after the kth step in the computation is defined as

$$C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)}, n_e^{(k)}),$$

where $n_i^{(k)}$ is the amount of spikes in neuron σ_i, for all $i = 1, 2, \ldots, m$, $n_e^{(k)}$ is the amount of spikes collected in the environment. Using this vector instead of the configuration vector in Definition 2 allows us to monitor the output of the system.

4 Matrix Representation for WSN P Systems

In this section, matrix representation for spiking neural P systems with weights (WSN P systems, for short) is investigated. Instead of counting spikes as in usual SN P systems, each neuron in WSN P systems contains a potential, which can be expressed by a computable real number. Each neuron fires when its potential is equal to the given threshold. The execution of a rule consumes part of the potential and produces a unit potential. This unit potential passes to neighboring neurons multiplied with the weights of synapses. In an SN P system with weights, the involved numbers – weights, thresholds, potential consumed by each rule – can be real (computable) numbers. Formally, the system is defined as follows.

Definition 7. *An SN P system with weights, of degree* $m \geq 1$*, is a construct of the form*

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

where:

1. $\sigma_1, \ldots, \sigma_m$ *are neurons of the form* $\sigma_i = (p_i, R_i), 1 \leq i \leq m$*, where:*
   a) $p_i \in \mathbb{R}_c$ *is the initial potential in* σ_i*;*
   b) R_i *is a finite set of rules of the form* $T_i/d_s \to 1$*,* $s = 1, 2, \ldots, n_i$ *for some* $n_i \geq 1$*, where* $T_i \in \mathbb{R}_c$*,* $T_i \geq 1$*, is the firing threshold potential of neuron* σ_i*, and* $d_s \in \mathbb{R}_c$ *with the restriction* $0 < d_s \leq T_i$*;*
2. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \times \mathbb{R}_c$ *are synapses between neurons, with* $i \neq j$*,* $w \neq 0$ *for each* $(i, j, w) \in syn, 1 \leq i, j \leq m$*;*
3. $in, out \in \{1, 2, \ldots, m\}$ *indicate the input and output neurons, respectively.*

The spiking rules are applied as follows. Assume that at a given moment, neuron σ_i has the potential equal to p. If $p = T_i$, then any rule $T_i/d_s \to 1 \in R_i$ can be applied. The execution of this rule consumes an amount of d_s of the potential (thus leaving the potential $T_i - d_s$) and prepares one unit potential (we also say a spike) to be delivered to all the neurons σ_j such that $(i, j, w) \in syn$. Specifically, each of these neurons σ_j receives a quantity of potential equal to w, which is added to the existing potential in σ_j. Note that w can be positive or negative, hence the potential of the receiving neuron is increased or decreased. The potential emitted by a neuron σ_i passes immediately to all neurons σ_j such that $(i, j, w) \in syn$, that is, the transition of potential takes no time. If a neuron σ_i spikes and it has no outgoing synapse, then the potential emitted by neuron σ_i is lost.

The main feature of this system is: (1) each neuron σ_i has only one fixed threshold potential T_i; (2) if a neuron has the potential equal to its threshold potential, then all rules associated with this neuron are enabled, and only one of them is non–deterministically chosen to be applied; (3) when a neuron spikes, there is always only one unit potential (a spike) emitted.

If neuron σ_i has the potential p such that $p < T_i$, then the neuron σ_i returns to the resting potential 0. If neuron σ_i has the potential p such that $p > T_i$, then the potential p keeps unchanged.

Summing up, if neuron σ_i has potential p and receives potential k at step t, then at step $t + 1$ it has the potential p', where:

$$p' = \begin{cases} k, & \text{if } p < T_i; \\ p - d_s + k, & \text{if } p = T_i \text{ and rule } T_i/d_s \to 1 \text{ is applied;} \\ \text{p+k}, & \text{if } p > T_i. \end{cases}$$

The configuration of the system is described by the distribution of potentials in neurons. Using the rules described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be applied. The result of a computation is the number of steps elapsed between the first two spikes sent by the output neuron to the environment during the computation.

Similar to Section 2, some vectors are defined to represent configurations and the application of rules.

Definition 8 (Configuration Vectors). *Let Π be a WSN P system with m neurons, the vector $C_0 = (p_1, p_2, \ldots, p_m)$ is called the* **initial configuration vector** *of Π, where p_i is the amount of the initial potential present in neuron σ_i, $i = 1, 2, \ldots, m$ before a computation starts.*

*In a computation, for any $k \in \mathbb{N}$, the vector $C_k = (p_1^{(k)}, p_2^{(k)}, \ldots, p_m^{(k)})$ is called the k***th configuration vector** *of the system, where $p_i^{(k)}$ is the amount of spikes in neuron σ_i, $i = 1, 2, \ldots, m$ after the kth step in the computation.*

Definition 9 (Spiking Vectors). *Let Π be a WSN P system with m neurons and n rules, and $C_k = (p_1^{(k)}, p_2^{(k)}, \ldots, p_m^{(k)})$ be the kth configuration vector of Π.*

Assume a total order $d : 1, \ldots, n$ *is given for all the* n *rules, so the rules can be referred as* $r_1, \ldots, r_n$. *A* **spiking vector** $s^{(k)}$ *is defined as follows:*

$$s^{(k)} = (r_1^{(k)}, r_2^{(k)}, \ldots, r_n^{(k)}),$$

where:

$$r_i^{(k)} = \begin{cases} 1, \textit{ if } r_i \textit{ in neuron } \sigma_j \textit{ is enabled (the potential } p_j^{(k)} \textit{ in neuron } \sigma_j \textit{ is equal} \\ \quad \textit{to its spiking threshold } t_j) \textit{ and the rule } r_i \textit{ is chosen and applied;} \\ 0, \textit{ otherwise.} \end{cases}$$

In particular, $s^{(0)} = (r_1^{(0)}, r_2^{(0)}, \ldots, r_n^{(0)})$ *is called the* **initial spiking vector**.

In WSN P systems, when the potential of a neuron is smaller than its spiking threshold, then this potential vanishes, the potential of the neuron is set to zero. In order to describe which neurons forget their potentials, *forgetting vector* is defined.

Definition 10 (Forgetting vector). *Let* Π *be a WSN P system with* m *neurons, and* $C_k = (p_1^{(k)}, p_2^{(k)}, \ldots, p_m^{(k)})$ *be the* k*th configuration vector of* Π. *A* **forgetting vector** $forg^{(k)}$ *is defined as follows:*

$$forg^{(k)} = (f_1^{(k)}, f_2^{(k)}, \ldots, f_m^{(k)}),$$

where:

$$f_i^{(k)} = \begin{cases} 1 & \textit{if the potential } p_i^{(k)} \textit{ in neuron } \sigma_i \textit{ is less than its spiking threshold } t_i; \\ 0 & \textit{otherwise.} \end{cases}$$

In particular, $forg^{(0)} = (f_1^{(0)}, f_2^{(0)}, \ldots, f_m^{(0)})$ *is called the* **initial forgetting vector**.

For a WSN P system, a *spiking transition matrix* is defined in order to store the information of the amount of potential consumed (or received) by each neuron when each rule is applied.

Definition 11 (Spiking Transition Matrix). *Let* Π *be a WSN P system with* m *neurons and* n *rules,* $d : 1, \ldots, n$ *a total order given for all the* n *rules. The* **spiking transition matrix** *of the system* Π, M_Π, *is defined as follows:*

$$M_\Pi = [a_{ij}]_{n \times m},$$

where:

$$a_{ij} = \begin{cases} -c, \textit{ if rule } r_i \textit{ is in neuron } \sigma_j \textit{ and it is applied consuming potential } c; \\ w, \textit{ if rule } r_i \textit{ is in neuron } \sigma_s \textit{ } (s \neq j \textit{ and } (s, j, w) \in syn) \\ \quad \textit{and it is applied;} \\ 0, \textit{ if rule } r_i \textit{ is in neuron } \sigma_s \textit{ } (s \neq j \textit{ and } (s, j, w) \notin syn, \textit{ for } w \in \mathbb{R}_c). \end{cases}$$

The following example illustrates how to get the matrix representation of a WSN P system.

Example 2. Let us consider a WSN P system $\Pi_1 = (\sigma_1, \sigma_1, \sigma_3, syn, out)$ that generates the set $\mathbb{N}$ of natural numbers excluding 1 and 2, where $\sigma_1 = (3, R_1)$, with $R_1 = \{1.5/1.5 \to 1, 1.5/1 \to 1\}$; $\sigma_2 = (2, R_2)$, with $R_2 = \{1/1 \to 1\}$; $\sigma_3 = (1.5, R_3)$, with $R_3 = \{1.5/1 \to 1\}$; $syn = \{(1,2,1),(1,3,-0.5),(2,1,1),(2,3,1.5),(3,1,-1.5),(3,2,-1)\}$; $out = 3$. Π_1 is also represented graphically in Figure 2. Note that in Figure 2, when the weight on a synapse is one, it is omitted.

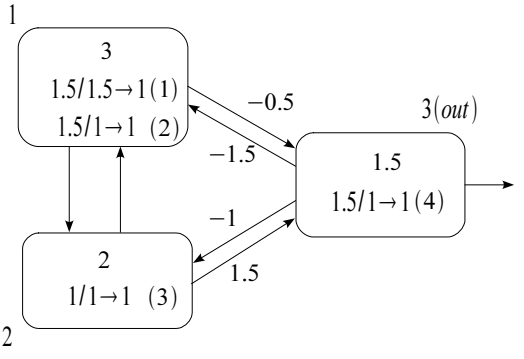


Fig. 2. A WSN P system Π_1 that generates the set $\mathbb{N} - \{1, 2\}$

As shown in Figure 2, a total order for the four rules is set. With this order, the rules can be referred as r_1, r_2, r_3, r_4. According to Definition 11, the spiking transition matrix M_{Π_1} for the WSN P system Π_1 is

$$M_{\Pi_1} = \begin{pmatrix} -1.5 & 1 & -0.5 \\ -1 & 1 & -0.5 \\ 1.5 & -1 & 1.5 \\ -1.5 & -1 & -1.5 \end{pmatrix} \tag{8}$$

The initial configuration is $C_0 = (3, 2, 1.5)$. The initial spiking vector is $(0, 0, 0, 1)$ by Definition 9 and the order of rules.

At step 1, only the output neuron σ_3 spikes, while the other two neurons σ_1, σ_2 maintain their potentials, because their potentials are greater than their corresponding firing thresholds. Neurons σ_1 and σ_2 receive potentials -1.5 and -1, respectively. After this step, the configuration vector becomes $C_1 = (1.5, 1, 0.5)$. At step 2, neurons σ_1 and σ_2 have potentials 1.5 and 1, respectively, which equal to their corresponding firing thresholds, hence both neurons σ_1 and σ_2 spike. Neuron σ_1 has two rules $r_1 : 1.5/1.5 \to 1$ and $r_2 : 1.5/1 \to 1$, and one of them is non-deterministically chosen. If rule r_1 is chosen to be applied, it consumes potential 1.5 and, at the same time, it receives 1.5 unit of potential from neuron σ_2. Hence, in this step, the net gain of potential in neuron σ_1 is 0 and at the next step neuron σ_1 still has potential 1.5. The net gain of potential in neuron

σ_2 is also 0 (one unit of potential is consumed by r_3 and one unit of potential is received from neuron σ_1), neuron σ_3 forgets its potential 0.5 (because it is less than the threshold 1.5), but gets another potential 0.5 from the other two neurons (receives potential -0.5 from σ_1 and potential 1 from σ_2). At the next step, the numbers of spikes in neurons σ_1, σ_2, and σ_3 are still 1.5, 1 and 0.5, respectively.

In order to denote the change of numbers of spikes in each neuron, a *transition net gain vector* is defined.

Definition 12 (Transition Net Gain Vector). *Let Π be a WSN P system with m neurons and n rules, and $C_k = (p_1^{(k)}, p_2^{(k)}, \ldots, p_m^{(k)})$ be the kth configuration vector of Π. The* **transition net gain vector** *at step k is defined as $NG^{(k)} = C_{k+1} - C_k$.*

In order to compute the transition net gain vector, the *Hadamard product* is used.

Definition 13 (Hadamard product). *Let A and B be $m \times n$ matrices. The Hadamard Product of A and B is defined by $[A \odot B]_{ij} = A_{ij}B_{ij}$ for all $1 \leq i \leq m$, $1 \leq j \leq n$.*

To avoid confusion, juxtaposition of matrices will imply the "usual" matrix multiplication, and "$\odot$" is used for the Hadamard product.

Lemma 2. *Let Π be a WSN P system with m neurons and n rules, $d : 1, \ldots, n$ be a total order for the n rules, M_Π the spiking transition matrix of Π, C_k the kth configuration vector, $s^{(k)}$ the spiking vector at step k, and $forg^{(k)}$ the forgetting vector at step k. Then the transition net gain vector at step k is*

$$NG^{(k)} = s^{(k)} \cdot M_\Pi - forg^{(k)} \odot C_k. \tag{9}$$

Proof. Assume that $M_\Pi = [a_{ij}]_{m \times n}$, $C_k = (p_1^{(k)}, p_2^{(k)}, \ldots, p_m^{(k)})$, $s^{(k)} = (r_1^{(k)}, r_2^{(k)}, \ldots, r_n^{(k)})$, $forg^{(k)} = (f_1^{(k)}, f_2^{(k)}, \ldots, f_m^{(k)})$, and $NG^{(k)} = (g_1, g_2, \ldots, g_m)$. Note that $r_i^{(k)}$ is a $\{0, 1\}$-value that identifies whether the rule r_i would be applied, $f_i^{(k)}$ is a $\{0, 1\}$-value that identifies whether the neuron σ_j would forget its potential. Thus, $\Sigma_{i=1}^n r_i^{(k)} a_{ij} - f_j^{(k)} p_j^{(k)}$ represents the total amount of potential obtained, consumed and forgotten by neuron σ_j at the kth step. Therefore, the net gain of neuron σ_j is $g_j = \Sigma_{i=1}^n r_i^{(k)} a_{ij} - f_j^{(k)} p_j^{(k)}$, for all $j = 1, 2, \ldots, m$. That is, $NG^{(k)} = s^{(k)} \cdot M_\Pi - forg^{(k)} \odot C_k$.

Theorem 2. *Let Π be a WSN P system with m neurons and n rules, $d : 1, \ldots, n$ be a total order for the n rules, M_Π the spiking transition matrix of Π, C_k the kth configuration vector, $s^{(k)}$ the spiking vector at step k, and $forg^{(k)}$ the forgetting vector at step k. Then the configuration C_{k+1} of Π can be obtained by*

$$C_{k+1} = C_k \ + \ s^{(k)} \cdot M_\Pi - forg^{(k)} \odot C_k. \tag{10}$$

Proof. This result follows directly from the preceding lemma.

In Example 2 shown in Figure 2, $C_0 = (3, 2, 1.5)$, at step 1 only the rule r_4 is applicable. As the potentials in all the neurons are higher than their thresholds, no neuron forgets its potential. Therefore, $s^{(0)} = (0, 0, 0, 1)$, $forg^{(0)} = (0, 0, 0)$, the next configuration can be obtained by

$$C_1 = (3, 2, 1.5) + (0, 0, 0, 1) \begin{pmatrix} -1.5 & 1 & -0.5 \\ -1 & 1 & -0.5 \\ 1.5 & -1 & 1.5 \\ -1.5 & -1 & -1.5 \end{pmatrix} - (0, 0, 0) \odot (3, 2, 1.5) \qquad (11)$$

That is, $C_1 = (2, 1, 0.5)$.

In the next step, rules r_1, r_3 can be applied, so the spiking vector is $(1, 0, 1, 0)$. Because the potential in neuron σ_3 is less than its threshold, neuron σ_3 forgets its potential, and the forgetting vector is $forg^{(0)} = (0, 0, 1)$. Hence, the next configuration is

$$C_2 = (2, 1, 0.5) + (1, 0, 1, 0) \begin{pmatrix} -1.5 & 1 & -0.5 \\ -1 & 1 & -0.5 \\ 1.5 & -1 & 1.5 \\ -1.5 & -1 & -1.5 \end{pmatrix} - (0, 0, 1) \odot (2, 1, 0.5) \qquad (12)$$

That is, $C_2 = (2, 1, 0.5)$.

This example clearly shows how matrix representation and operation can describe the computation of the system. Such matrix representation is useful for the simulation of the system in computer.

5 Conclusions and Remarks

In this work, an algebraic representation for SN P systems is introduced. For every SN P system without delay, configuration vectors are defined to represent the number of spikes in each neuron; spiking vectors are used to denote which rules will be applied; a spiking transition matrix is used to describe the skeleton of system. Such algebraic representation is also extended for another variant of SN P systems – WSN P systems.

It is not difficult to see that such matrix representation is also suitable for other variants of SN P systems, such as asynchronous SN P systems [1] and SN P systems with exhaustive use of rules [4]. The spiking transition matrix is related to the structure of system only, so the elements of the matrix are determined initially. During the computation of a system, it is only needed to decide the spiking vector by checking the current configuration vector and the regular expressions of rules. In general, such algebraic representation is easy to be programmed for computer simulation.

The systems considered in this paper have no delay, which corresponds to the biological feature that neurons have refractory time. It is open how to represent the computations of SN P systems with delay by matrices.

Acknowledgements

The work of X. Zeng and L. Pan was supported by National Natural Science Foundation of China (61033003 and 30870826) and the Fundamental Research Funds for the Central Universities (2010ZD001). The work of H. Adorna is supported by Engineering Research and Development for Technology of the DOST, Philippines. The work of M.A. Martínez-del-Amor and M.J. Pérez-Jiménez is supported by the project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds, and the "Proyecto de Excelencia con Investigador de Reconocida Valía" of the Junta de Andalucía under grant P08-TIC04200.

References

1. Cavaliere, M., Egecioglu, O., Ibarra, O.H., Woodworth, S., Ionescu, M., Păun, G.: Asynchronous Spiking Neural P Systems. Theoretical Computer Science 410, 2352–2364 (2009)
2. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Searching Previous Configurations in Membrane Computing. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 301–315. Springer, Heidelberg (2010)
3. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae 71(2–3), 279–308 (2006)
4. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems with Exhaustive Use of Rules. International Journal of Unconventional Computing 3, 135–154 (2007)
5. Nelson, J.K., McCormac, J.C.: Structural Analysis: Using Classical and Matrix Methods, 3rd edn. Wiley, Chichester (2003)
6. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
7. Păun, G.: Membrane Computing – An Introduction. Springer, Berlin (2002)
8. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
9. Wang, J., Hoogeboom, H.J., Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Weights. Neural Computation 22(10), 2615–2646 (2010)
10. The P System Web Page, http://ppage.psystems.eu

Author Index