Nineteenth Brainstorming Week on Membrane Computing

Sevilla, January 24 – 27, 2023

David Orellana-Martín
Agustín Riscos-Núñez
Gheorghe Păun
Editors

Nineteenth Brainstorming Week on Membrane Computing

Sevilla, January 24 – 27, 2023

David Orellana-Martín
Agustín Riscos-Núñez
Gheorghe Păun
Editors

RGNC REPORT 1/2023

Research Group on Natural Computing
Universidad de Sevilla

Sevilla, 2023

ISBN: 978-84-09-54839-2

Printed by: Artes Gráficas Moreno, S.L.
`http://www.agmoreno.net/`

Preface

The Nineteenth Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from January 24 to 27, 2023, hosted by the Research Group on Natural Computing (RGNC) from the Department of Computer Science and Artificial Intelligence of Universidad de Sevilla. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and all the next editions have been taking place in Sevilla since then, always at the end of January and/or at the beginning of February.

In the style of previous meetings in this series, was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation. Several "provocative" talks were delivered, mainly devoted to open problems, research topics, announcements, conjectures waiting for proofs, or ongoing research works in general (involving both theory and applications). Joint work sessions were scheduled on the afternoons to allow for collaboration among the about 30 participants – see the list in the end of this preface.

In the first day of the meeting, the third SCORE Workshop on Membrane Computing were co-located, where the members of the SCORE Unit of Excellence (`https://score.us.es/`) were introduced to the P community.

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

A selection of papers from this volume will be considered for publication in the *International Journal of Neural Systems*, published by World Scientific (`https://www.worldscientific.com/worldscinet/ijns`).

Other papers elaborated during the 2023 edition of BWMC will be submitted to other journals or to suitable conferences. The reader interested in the final version of these papers is advised to check the current bibliography of membrane computing available in the domain website `http://ppage.psystems.eu`.

***

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

1. Artiom Alhazov, Institute of Mathematics and Computer Science of Academy of Sciences of Moldova, Moldova
   artiom@math.md
2. José A. Andreu-Guzmán, Universidad de Sevilla, Spain
   josandguz@gmail.com
3. Julien Caselmann, Humboldt-Universität zu Berlin
   caselmaj@informatik.hu-berlin.de
4. Lúdek Cienciala, Silesian University in Opava, Czech Republic
   ludek.cienciala@fpf.slu.cz
5. Lucie Ciencialová, Silesian University in Opava, Czech Republic
   lucie.ciencialova@fpf.slu.cz
6. Michael Dinneen, University of Auckland, New Zealand
   mjd@cs.auckland.ac.nz
7. Rudolf Freund, Technological University of Vienna, Austria
   rudi@emcc.at
8. Marian Gheorghe, University of Bradford, United Kingdom
   m.gheorghe@bradford.ac.uk
9. Carmen Graciani, Universidad de Sevilla, Spain
   cgdiaz@us.es
10. Florentin Ipate, University of Bucharest, Romania
    florentin.ipate@unibuc.ro
11. Sergiu Ivanov, IBISC, Université Évry, Université Paris-Saclay, France
    sergiu.ivanov@univ-evry.fr
12. Anna Kuczik, University of Debrecen, Hungary
    kuczik.anna@inf.unideb.hu
13. Miguel A. Martínez-del-Amor, Universidad de Sevilla, Spain
    mdelamor@us.es
14. Radu Nicolescu, University of Auckland, New Zealand
    r.nicolescu@auckland.ac.nz
15. David Orellana-Martín, Universidad de Sevilla, Spain
    dorellana@us.es
16. Gheorghe Păun, Romanian Academy, Romania
    curteadelaarges@gmail.com
17. Ignacio Pérez-Hurtado, Universidad de Sevilla, Spain
    perezh@us.es
18. Mario de J. Pérez-Jiménez, Universidad de Sevilla, Spain
    marper@us.es
19. Mihail-Iulian Pleșa, University of Bucharest, Romania
    mihail-iulian.plesa@s.unibuc.ro

20. Antonio Ramírez-de-Arellano, Universidad de Sevilla
aramirezdearellano@us.es
21. Damien Regnault, IBISC, Université Évry, Université Paris-Saclay, France
damien-regnault@univ-evry.fr
22. Agustín Riscos-Núñez, Universidad de Sevilla, Spain
ariscosn@us.es
23. José Antonio Rodríguez-Gallego, Universidad de Sevilla, Spain
jrodriguez14@us.es
24. Álvaro Romero-Jiménez, Universidad de Sevilla, Spain
romero.alvaro@us.es
25. José María Sempere-Luna, Universitat Politècnica de València, Spain
jsempere@dsic.upv.es
26. Luis Valencia-Cabrera, Universidad de Sevilla, Spain
lvalencia@us.es
27. György Vaszil, University of Debrecen, Hungary
vaszil.gyorgy@inf.unideb.hu
28. Claudio Zandron, University of Milano-Bicocca, Italy
claudio.zandron@unimib.it
29. Gexiang Zhang, Chengdu University of Information Technology, China
gexiangzhang@gmail.com

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

The meeting was partially supported from various sources: (i) FEDER/Junta de Andalucía – Paidi 2020/ _Proyecto (P20_00486), (ii) VII Plan Propio, *Vicerrectorado de Investigación de la Universidad de Sevilla*, (iii) Department of Computer Science and Artificial Intelligence from *Universidad de Sevilla*, and (iv) the Research Institute of Computer Engineering (I3US) of the University from *Universidad de Sevilla.*

The Editors
(Jul 2023)

Contents

Simple P Systems with Prescribed Teams of Sets of Rules

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

[1] State University of Moldova,
Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[3] IBISC, Univ. Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

Summary. In this paper we consider simple P systems with prescribed teams of sets of rules, with the application of the rule sets in the teams probably depending on some given condition, as well as, in the general case, the different sets of rules in a prescribed team working in different derivation modes, whereas in homogeneous systems for all sets of rules the same derivation mode comes into action.

We prove some general results, for example, how with such simple P systems with prescribed teams of sets of rules we can simulate label controlled P systems, where only rules with the same label can be applied, as well as how simple purely catalytic P systems can be mimicked by simple P systems with prescribed teams of sets of non-cooperative rules with all sets of rules working in the sequential derivation mode and how simple catalytic P systems can be mimicked by simple P systems with prescribed teams of sets of non-cooperative rules with some sets of rules working in the sequential derivation mode and only one working in the maximally parallel derivation mode.

Computational completeness of these simple P systems with prescribed teams of sets of non-cooperative rules therefore immediately follows from the well-known results for simple catalytic and purely catalytic P systems, respectively. On the other hand, homogeneous simple P systems with prescribed teams of sets of non-cooperative rules with all teams working in the maximally parallel derivation mode have the same computational power as $ET0L$ systems used for multisets.

Keywords: applicability condition, computational completeness, $ET0L$ systems, P systems, prescribed teams

1 Introduction

A quarter of a century ago, membrane (P) systems were introduced in [12] as a multiset-rewriting model of computing inspired by the structure – the hierarchical membrane structure – and the functioning of the living cell – with the molecules/objects evolving in parallel. Since then, this area of biologically motivated computing has emerged in a fascinating way. A lot of interesting theoretical models have been developed by scientists all over the world, many of them already documented in two textbooks, see [13] and [14]. For actual information, we refer to the *P systems webpage* [16] as well as to the issues of the *Bulletin of the International Membrane Computing Society* and of the *Journal of Membrane Computing.*

P systems traditionally operate on multisets of objects, hence, the power of non-cooperative rules (even) when working in the maximally parallel derivation mode is rather restricted; for example, the multiset language $\{b^{2^n} \mid n \in \mathbb{N}\}$ cannot be obtained with non-cooperative rules by halting computations. Therefore, one of the fundamental questions which has attracted a lot of attention in the area of P systems is, how variants of different ways of cooperation of the rules and various control mechanisms affect the computational power. For example, allowing for cooperative rules rather easily boosts the power of specific variants of P systems to computational completeness.

One of the well-known control mechanisms forcing some rules to only be applied together (in the sequential derivation mode) are matrix grammars, in which the rules are grouped into sequences, which in the given order must be applied one after another. A less strict variant where the rules in a set of rules called *prescribed teams* can be applied in any order was introduced in [6]. In [4], such prescribed teams are working on different objects.

In contrast to the original model, in which the rules of a team can be applied together only sequentially, we here consider a *team* as a set of sets of rules, where each set of rules has assigned (i) its own applicability condition and (ii) its own derivation mode in which the rules in this set have to be applied, and based on one of these teams a suitable multiset of rules to be applied to the underlying configuration is constructed.

In the model of (*simple*, i.e., only one membrane region is considered) P systems with prescribed teams of sets of rules, the application of a team means applying each set of rules in the chosen team to be used in the derivation mode assigned to the set in this team, provided the applicability condition based on the features of the underlying configuration is fulfilled. In *internally homogenous* systems, all sets of rules in a team have assigned the same derivation mode, whereas in *globally homogenous* systems all teams have assigned the same derivation mode for all sets of rules in the teams. In this paper, we mainly focus on the sequential and the maximally parallel derivation mode; investigations with other derivation modes, as, for example considered in the formal framework for static P systems, see [9], or others then defined in [5, 2, 3, 1], we leave for future research.

One obvious result we are going to prove is that globally homogenous simple P systems working in the maximally parallel derivation mode for the teams of sets of non-cooperative rules have the same computational power as $ET0L$ systems, i.e., extended tabled Lindenmayer systems. Simple P systems with prescribed teams of sets of rules can simulate label controlled P systems, where only rules with the same label can be applied. Moreover, simple purely catalytic P systems can be mimicked by simple P systems with prescribed teams of sets of non-cooperative rules with the sets of rules working in the sequential derivation mode. For the simulation of catalytic P systems, one additional set working in the maximally parallel derivation mode is needed. Computational completeness of these simple P systems with prescribed teams of sets of non-cooperative rules therefore can immediately be inferred from the well-known results for simple catalytic and purely catalytic P systems, respectively. Furthermore, using sets of symbols as permitting and forbidden context conditions for the sets of rules in the teams allows for an easy direct simulation of register machines, either with using non-cooperative rules or else insertion and deletion rules.

2 Definitions

The cardinality of a set M is denoted by $|M|$. For further notions and results in formal language theory we refer to textbooks like [7] and [15].

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

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

The family of regular, context-free, and recursively enumerable string languages is denoted by $\mathcal{L}(REG)$, $\mathcal{L}(CF)$, and $\mathcal{L}(RE)$, respectively. As $Ps\mathcal{L}(REG) =$

$Ps\mathcal{L}(CF)$, in the area of multiset rewriting $\mathcal{L}(CF)$ plays no role at all, and in the area of membrane computing we often only get characterizations of $Ps\mathcal{L}(REG)$ and $Ps\mathcal{L}(RE)$ or else $Ps\mathcal{L}(ET0L)$, where $\mathcal{L}(ET0L)$ denotes the family of languages generated by extended tabled Lindenmayer systems ($ET0L$ systems).

For further notions and results in formal language theory we refer to textbooks like [7] and [15].

2.1 Register Machines

Register machines are well-known universal devices for computing on (or generating or accepting) sets of vectors of natural numbers. The following definitions and propositions are given as in [1].

Definition 1. *A* register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- m *is the number of registers,*
- P *is the set of instructions bijectively labeled by elements of* B*,*
- $l_0 \in B$ *is the initial label, and*
- $l_h \in B$ *is the final label.*

The instructions of M *can be of the following forms:*

- $p : (ADD(r), q, s)$*;* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
  Increase the value of register r *by one, and non-deterministically jump to instruction* q *or* s*.*
- $p : (SUB(r), q, s)$*;* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
  If the value of register r *is not zero then decrease the value of register* r *by one (*decrement *case) and jump to instruction* q*, otherwise jump to instruction* s *(*zero-test *case).*
- $l_h : HALT$*.*
  Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.* M *is called* deterministic *if the ADD-instructions all are of the form* $p : (ADD(r), q)$*.*

Throughout the paper, B_{ADD} *denotes the set of labels of ADD-instructions* $p : (ADD(r), q, s)$ *of arbitrary registers* r*, and* $B_{SUB(r)}$ *denotes the set of labels of all SUB-instructions* $p : (SUB(r), q, s)$ *of a decrementable register* r*. Moreover, for any* $p \in B \setminus \{l_h\}$*,* $Reg(p)$ *denotes the register affected by the ADD- or SUB-instruction labeled by* p*; for the sake of completeness, in addition* $Reg(l_h) = 1$ *is taken.*

In the *accepting* case, a computation starts with the input of an l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled by l_0); it terminates with reaching the $HALT$-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled by l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *computing* case, a computation starts with the input of an l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled by l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

For useful results on the computational power of register machines, we refer to [11]; for example, to prove our main theorem, we need the following formulation of results for register machines generating or accepting recursively enumerable sets of vectors of natural numbers with k components or computing partial recursive relations on vectors of natural numbers:

Proposition 1. *Deterministic register machines can accept any recursively enumerable set of vectors of natural numbers with l components using precisely $l+2$ registers. Without loss of generality, we may assume that at the end of an accepting computation all registers are empty.*

Proposition 2. *Register machines can generate any recursively enumerable set of vectors of natural numbers with k components using precisely $k+2$ registers. Without loss of generality, we may assume that at the end of a generating computation the first two registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

Proposition 3. *Register machines can compute any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

In all cases it is essential that the output registers never need to be decremented.

2.2 Extended Tabled Lindenmayer Systems

An *extended tabled Lindenmayer system* (an *ET0L system* for short) is a construct

$$G = (V, \Sigma, T_1, \dots, T_n, A) \quad \text{where}$$

- V is a set of objects;
- $\Sigma \subseteq V$ is a set of *terminal objects*;
- T_j, $1 \leq i \leq n$, called *tables* are finite sets of non-cooperative rules over V, i.e., of the form $a \to u$ with $a \in V$ and $u \in V^*$;
- $A \in V^+$ is the axiom.

A computation in the $ET0L$ system G starts with the axiom A; then, in each computation step, a table T_j is chosen and the rules in T_j are applied to the current configuration in a parallel way. The language generated by G is the set of all terminal strings in Σ^* obtained in that way from the axiom A, i.e.,

$$L(G) = \{w \in \Sigma^* \mid A \Longrightarrow^* w\}.$$

$ET0L$ systems can also be considered as computing models working on multisets instead of strings, i.e., the axiom A is the initial multiset and the configurations are multisets on which the non-cooperative rules in the tables work in parallel. In the following, such $ET0L$ systems working on multisets will be denoted as $mET0L$ systems. Obviously, we have $\mathcal{L}(mET0L) = Ps\mathcal{L}(ET0L)$.

Remark 1. As a technical detail we mention that many authors require every table to contain at least one rule for every object in V. We observe that incomplete tables missing a rule for some $x \in V$ can easily be made complete by adding the unit rules $x \to x$ for all $x \in V$ for which so far no rule is already present in the table.

3 Simple P Systems

Taking into account the well-known flattening process, which means that computations in a P system with an arbitrary (static) membrane structure can be simulated in a P system with only one membrane, e.g., see [8], in this paper we only consider simple P systems, i.e., with the simplest membrane structure of only one membrane region:

Definition 2. *A* simple P system *is a construct*

$$\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, \delta)$$

where

- *V is the alphabet of* objects*;*
- *$\mathcal{C} \subseteq V$ is the alphabet of* catalysts*;*
- *$\Sigma \subseteq (V \setminus \mathcal{C})$ is the alphabet of* terminal objects*;*
- *$w \in V^\circ$ is the multiset of objects initially present in the membrane region;*
- *$\mathcal{R}$ is a finite set of* evolution rules *over V; these evolution rules are multiset rewriting rules $u \to v$ with $u, v \in V^\circ$;*
- *δ is the* derivation mode*.*

A *catalytic rule* is of the form $ca \to cv$, a *non-cooperative* rule is of the form $a \to v$, where c is a catalyst, a is an object from $V \setminus \mathcal{C}$, and v is a string from $(V \setminus \mathcal{C})^*$. A simple P system only using catalytic and non-cooperative rules is called *catalytic*, and it is called *purely catalytic* if only catalytic rules are used. The *type* of a (simple) P system only using non-cooperative rules is abbreviated by *ncoo*, the types of catalytic and purely catalytic P systems are abbreviated by *cat* and *pcat*, respectively.

The multiset in the single membrane region of Π constitutes a *configuration* of the P system. The *initial configuration* is given by the initial multiset w; in case of accepting or computing P systems the input multiset w_0 is assumed to be added to w, i.e., the initial configuration then is ww_0.

A transition between configurations is governed by the application of the evolution rules, which is done in the given derivation mode δ. The application of a rule $u \to v$ to a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. Observe that each catalyst can be used (at most) once in every derivation step.

If no catalysts are used, we omit $\mathcal{C}$ and simply write $\Pi = (V, \Sigma, w, \mathcal{R}, \delta)$.

3.1 Variants of Derivation Modes

Given a P system $\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, \delta)$, the set of multisets of rules applicable to a configuration C is denoted by $Appl(\Pi, C)$.

The set of all multisets of rules applicable to a given configuration can be restricted by imposing specific conditions, thus yielding the following basic derivation modes (for example, see [9] for formal definitions):

- asynchronous mode (abbreviated *asyn*): at least one rule is applied;
- sequential mode (*sequ*): only one rule is applied;
- maximally parallel mode (*max*): a non-extendable multiset of rules is applied;
- maximally parallel mode with maximal number of rules (max_{rules}): a non-extendable multiset of rules of maximal possible cardinality is applied;
- maximally parallel mode with maximal number of objects ($max_{objects}$): a non-extendable multiset of rules affecting as many objects as possible is applied.

If $Appl(\Pi, C)$ is not empty, this set equals the set $Appl(\Pi, C, asyn)$ of multisets of rules applicable in the *asynchronous derivation mode* (abbreviated *asyn*).

In [5], these derivation modes are restricted in such a way that each rule can be applied at most once, thus yielding the set modes *sasyn*, *smax*, $smax_{rules}$, and $smax_{objects}$ (the sequential mode is already a set mode by definition).

In this paper we shall restrict ourselves to the derivation modes *sequ* and *max*:

The set $Appl(\Pi, C, sequ)$ denotes the set of multisets of rules applicable in the *sequential derivation mode* (abbreviated *sequ*), where in each derivation step exactly one rule is applied.

The standard parallel derivation mode used in P systems is the *maximally parallel derivation mode* (*max* for short), in which only non-extendable multisets of rules can be applied:

$$\begin{aligned} Appl(\Pi, C, max) = \{ & R \in Appl(\Pi, C) \mid \\ & \text{there is no } R' \in Appl(\Pi, C) \\ & \text{such that } R' \supset R\}. \end{aligned}$$

For some new variants of derivation modes we refer to [2, 3].

3.2 Computations in a P System

The P system continues with applying multisets of rules according to the given derivation mode until there remain no applicable rules in the single region of Π, i.e., as usual, with all these variants of derivation modes as defined above, we consider *halting computations.*

We may generate or accept or even compute functions or relations. The inputs/outputs may be multisets or strings, defined in the well-known way. When the system *halts*, in case of computing with multisets we consider the number of objects from Σ contained in the membrane region at the moment when the system halts as the *result* of the underlying computation of Π.

We would like to emphasize that as results we only take the objects from the terminal alphabet Σ, especially the catalysts are not counted to the result of a computation. On the other hand, with all the proofs given in this paper, except for the catalysts – if any – no other "garbage" remains in the membrane region at the end of a halting computation, i.e., we could even omit Σ.

3.3 (Simple) P Systems With Label Control

We may extend the model of a simple P system to the model of a *simple P system with label control*

$$\Pi = (V, \mathcal{C}, \Sigma, w, B, \mathcal{R}, \delta)$$

by labelling each rule in $\mathcal{R}$ by an element from a set of labels B. Then in any derivation step only rules labeled by the same label $r \in B$ are allowed to be used together. Such controlled P systems were investigated in [10].

Example 1. Consider the simple P system of type *ncoo* without catalysts

$$\Pi = (V = \{a, b\}, \Sigma = \{a\}, w = b, B = \{1, 2\}, \mathcal{R} = \{1 : b \to bb, 2 : b \to a\}, max)$$

with the two labels 1 and 2 in B as well as the labeled rules $1 : b \to bb$ and $2 : b \to a$ in $\mathcal{R}$.

Applying rule $1 : b \to bb$ $n \geq 0$ times we obtain b^{2^n}; by applying the second rule $2 : b \to a$ we finally obtain the terminal multiset a^{2^n}. Hence, $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$, a multiset language which cannot be obtained by a simple P system of type *ncoo* without additional control mechanism.

4 Simple P Systems with Prescribed Teams of Sets of Rules

We now consider a new model of simple P systems, where in one derivation step specific sets of rules – called *teams* – are applied in their assigned derivation mode. As usual, we start with a finite multiset of objects until no such team can be applied any more.

Definition 3. *A* simple P system with prescribed teams of sets of rules *– a* PPT system *for short – is a construct*

$$\Pi = (V, \Sigma, P, T_1, \dots, T_n, A) \text{ where}$$

- V *is a set of objects;*
- $\Sigma \subseteq V$ *is a set of* terminal objects*;*
- P *is a finite set of* multiset rules*, i.e., each rule is the form* $u \to v$ *with* $u \in V^*$ *and* $v \in V^+$*;*
- *each* prescribed team T_j*,* $1 \leq i \leq n$*, is a finite set of sets of rules from* P *together with the associated derivation mode* δ_j *and possibly some applicability condition* K_j*, i.e.,* $T_j = (\{(K_{j,i}, R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\})$*, where the* $R_{j,i} \subseteq P$ *are finite sets of rules from* P*;*
- $A \in V^\circ$ *is a finite multiset of initial objects from* V*.*

As usual, a rule $p \in P,\ p = u \to v$, is called *applicable* to a *configuration*, i.e., an object $x \in V^\circ$, if and only if u is a subset of x. The set of all rules applicable to x is denoted by $Appl(\Pi, x)$

The number n of teams is called the *degree* of Π. $|T_j|$ is called the *size* of the prescribed team T_j. If all prescribed teams have at most size s, then Π is called a *PPT system of size s*. If the number of rules in the sets of rules is at most m, then Π is called a *PPT system of rule size m*. Π is called a *PPT system of type* (n, s, m), if it is of degree n, size s, and rule size m. Moreover, if all rules in the sets of rules in the teams are of a specific type α (for example *ncoo*), we call Π a *PPT system of type* $(\alpha; n, s, m)$.

The family of sets of multisets generated/accepted by PPT system of type $(\alpha; n, s, m)$ is denoted by $\mathcal{L}(PPT_{gen}(\alpha; n, s, m))/\mathcal{L}(PPT_{acc}(\alpha; n, s, m))$. Any of the parameters n, s, m can be replaced by $*$, if the number cannot be bounded; α can also be omitted.

As derivation modes, we will restrict ourselves to the sequential derivation mode *sequ* and the maximally parallel derivation mode *max*.

The conditions $K_{j,i}$ in the most general case can be any computable/recursive features of the underlying configuration. Here we essentially will consider random context conditions, i.e., $K_{j,i} = (P_{j,i}, Q_{j,i})$, where $P_{j,i}$ and $Q_{j,i}$ are finite sets of multisets over V; $P_{j,i}$ is the set of *permitting contexts* and $Q_{j,i}$ is the set of *forbidden contexts.* The random context condition $K_{j,i} = (P_{j,i}, Q_{j,i})$ is fulfilled by a multiset x if and only if x contains each multiset in $P_{j,i}$, but none of the multisets in $Q_{j,i}$. If no conditions $K_{j,i}$ are specified, we simply write $T_j = \{(R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\}$.

If different derivation modes appear in a team, the whole PPT system is called *non-homogenous*. The system is called *locally homogenous*, if for all teams, the sets of rules in the team all are applied in the same derivation mode δ_j, and we write $T_j = (\{R_{j,i} \mid 1 \leq i \leq n_j\}, \delta_j)$, $1 \leq j \leq n$. Finally, the system is called *globally homogenous* if the derivation mode is the same δ for all T_j, and we only write $T_j = \{R_{j,i} \mid 1 \leq i \leq n_j\}$ and specify δ by writing $\Pi = (V, \Sigma, P, T_1, \dots, T_n, \delta, A)$.

Computations in a PPT system

Given a prescribed team of sets of rules

$$T_j = (\{(K_{j,i}, R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\})$$

with the derivation modes $\{\delta_{j,i} \mid 1 \leq i \leq n_j\} \subseteq \{sequ, max\}$, a derivation step with T_j on the configuration x can be carried out in the following way:

1. we choose a multiset of rules $R \in Appl(\Pi, x)$; the multiset of objects binded by R is denoted by $Bind(R, x)$, the multiset of objects in $x \setminus Bind(R, x)$ is denoted by $Idle(R, x)$;
2. we now for all $1 \leq i \leq n_j$ check whether x fulfills the applicability conditions $K_{j,i}$;
3. each rule in R must be assigned to one of the sets $R_{j,i}$ for which the applicability condition $K_{j,i}$ is fulfilled, yielding the multiset of rules $R'_{j,i}$; the multiset of objects binded by the rules in $R'_{j,i}$ is denoted by $Bind(R, R'_{j,i}, K)$;
4. for $\delta_{j,i} = max$ we now check that $R'_{j,i}$ cannot be extended by using an additional rule from $R_{j,i}$ on objects from $Idle(R, x)$;
5. for $\delta_{j,i} = sequ$ we check whether $|R'_{j,i}| = 1$; if not, then we have to check that no rule from $R_{j,i}$ can be applied to objects from $Idle(R, x)$;
6. if all checks from above have been passed correctly, the multiset of rules R can be applied to the current configuration x.

We emphasize that the rule sets in a team compete for the objects available in the underlying configuration, but at the end each set of rules for itself makes its part of the transition from the underlying configuration to the next configuration in a correct way according to its assigned derivation mode, as no idle object could be binded by an additional rule. Moreover, we observe that a set of rules $R_{j,i}$ from T_j can only be chosen if the applicability condition $K_{j,i}$ is fulfilled by x. Finally, a team T_j can only be applied if the multiset of rules obtained by the procedure described above is not empty.

As some variant of the general model we may also consider prescribed teams of sets of rules for which the applicability conditions $K_{j,i}$ are the same for all $1 \leq i \leq n_j$, i.e., just one condition K_j, and then we write

$$T_j = (K_j, \{(R_{j,i}, \delta_{j,i}) \mid 1 \leq i \leq n_j\})$$

and can simplify the procedure for applying T_j by first checking that the current configuration fulfills K_j.

As a first example we show how label control can easily be simulated by teams of sets of rules:

Example 2. Consider the globally homogenous PPT system of type *ncoo*

$$\Pi = (V = \{a, b\}, \Sigma = \{a\}, P, T_1, T_2, max, b) \quad \text{where}$$

$P = \{b \to bb, b \to a\}$, $T_1 = \{\{b \to bb\}\}$, and $T_2 = \{\{b \to a\}\}$. Π is a globally homogenous PPT system of type $(ncoo; 2, 1, 1)$.

Applying team T_1, i.e., the rule $b \to bb$, in the maximally parallel way $n \geq 0$ times we obtain b^{2^n}; by applying the second team T_2, i.e., the rule $b \to a$, in the maximally parallel way once, we finally obtain the terminal multiset a^{2^n} as in Example 1. Hence, we conclude $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$ as well as

$$\{a^{2^n} \mid n \geq 0\} \in \mathcal{L}(gh(max)PPT_{gen}(ncoo; 2, 1, 1)),$$

with the prefix $gh(max)$ indicating that we consider globally homogenous PPT systems working in the derivation mode *max*.

5 PPT Systems Simulating P Systems With Label Control

As can already be guessed by looking at Example 2, PPT systems can easily simulate P systems with label control – *without catalysts* – which are working in the derivation mode *max* by putting the rules with the same labels into one team:

Theorem 1. *Every P systems with label control Π, without catalysts, and working in the derivation mode max, can be simulated by a PPT system of type $(n, 1, *)$, where n is the number of different labels for the rules in Π.*

Proof. Given a simple P system with label control, without catalysts,

$$\Pi = (V, \Sigma, w, B, \mathcal{R}, max)$$

where each rule in $\mathcal{R}$ is labelled by an element from a set of labels B, $B = \{l_j \mid 1 \leq j \leq n\}$, we construct a globally homogenous PPT system Ψ of degree n and size 1, simulating (the computations of) Π:

$$\Psi = (V, \Sigma, P, T_1, \ldots, T_n, max, w)$$

where we define

$$P = \{p \mid l_j : p \in \mathcal{R},\ 1 \leq j \leq n\}$$

as well as the teams T_j, $1 \leq j \leq n$, as follows:

$$T_j = \{\{p \mid l_j : p \in \mathcal{R}\}\}$$

By definition, the size of T_j is 1, whereas the number of rules in the single set of rules in a team can be arbitrarily large.

We observe that applying the set of rules in in the team T_j in Ψ in the maximally parallel way has the same effect as applying exactly the rules with label l_j in Π in the maximally parallel way. □

6 PPT Systems Simulating $mET0L$ Systems

We now show that the computational power of $mET0L$ systems equals the computational power of globally homogenous PPT systems of type *ncoo* without applicability conditions working in the derivation mode *max*.

Theorem 2. *Every $mET0L$ system with n tables can be simulated by a globally homogenous PPT system of type ncoo, degree n, and size 1 without applicability conditions working in the derivation mode max.*

Proof. The $mET0L$ system $G = (V, \Sigma, T_1', \dots, T_n', A)$ can be simulated by the globally homogenous PPT system of type *ncoo*, degree n, and size 1 without applicability conditions working in the derivation mode *max*

$$\Pi = (V, \Sigma, P, T_1, \dots, T_n, max, A)$$

where we simply take

$$T_j = \{T_j' \setminus \{a \to a \mid a \in \Sigma\}\}, \ \ 1 \leq j \leq n,$$

i.e., the work of the table T_j' is simulated by the single set of rules in the team T_j of the PPT system Π.

We observe that we have to exclude the unit rules $a \to a$ for the terminal symbols $a \in \Sigma$, from the sets of rules T_j', $1 \leq j \leq n$, in order to ensure that Π halts as soon as a terminal configuration (i.e., a configuration only containing terminal symbols) has been reached. Finally, we mention that every (useless) team T_j of the form $\{\emptyset\}$ is to be omitted. □

In order to show the inverse inclusion, we need the following lemma:

Lemma 1. *Every globally homogenous PPT system of degree n and size k without applicability conditions working in the derivation mode max can be simulated by a globally homogenous PPT system of degree n and size 1 without applicability conditions working in the derivation mode max.*

Proof. The globally homogenous PPT system of degree n and size k without applicability conditions working in the derivation mode *max*

$$\Pi = (V, \Sigma, P, T_1, \dots, T_n, max, A)$$

with $T_j = \{R_{j,i} \mid 1 \leq j \leq n_j\}$, $1 \leq j \leq n$, can be simulated by the corresponding globally homogenous PPT system of degree n and only size 1 without applicability conditions working in the derivation mode *max*

$$\Pi' = (V, \Sigma, P, T_1', \dots, T_n', max, A)$$

where we take $T_j' = \{\{y \mid y \in R_{j,i}, \ 1 \leq i \leq n_j\}\}$. We observe that by definition the rules in the $R_{j,i}$ work in parallel on the underlying configurations in the same way if they are grouped in the $R_{j,i}$ or just in one set of rules $\{y \mid y \in R_{j,i}, \ 1 \leq i \leq n_j\}$. We observe that Π' again is of degree n, but only of size 1. □

Based on this lemma, we now can show how a globally homogenous PPT system of type *ncoo*, degree n without applicability conditions working in the derivation mode *max* can be simulated by an $mET0L$ system with n tables:

Theorem 3. *Every globally homogenous PPT system of type ncoo, degree n, and size k without applicability conditions working in the derivation mode max can be simulated by an $mET0L$ system with n tables.*

Proof. According to Lemma 1, without loss of generality we may assume that the size k is only one. Hence, we may start with a globally homogenous PPT system of type *ncoo*, degree n, and size 1 without applicability conditions working in the derivation mode *max*

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, max, A)$$

where for the teams T_j we have $T_j = \{T'_j\}$, $1 \leq j \leq n$, with T'_j being a set of non-cooperative rules.

Then the $mET0L$ system

$$G = (V, \Sigma, T'_1, \ldots, T'_n, A)$$

simulates the (computations of the) PPT system Π, as the work of the table T'_j simulates the application of the team T_j of the PPT system Π with the single set of rules T'_j.

As a technical detail we mention that the tables T'_j have to be extended by unit rules $x \to x$ for every $x \in V$ for which no rule is already present in it, in order to fulfill the requirement for $ET0L$ systems as already discussed in Remark 1. $\square$

In sum, we have shown the following result (where $gh(max)PPT(ncoo)$ denotes the globally homogenous PPT systems of type *ncoo* working in the derivation mode *max*):

Theorem 4. $\mathcal{L}(mET0L) = \mathcal{L}(gh(max)PPT(ncoo))$.

7 PPT Systems Simulating [Purely] Catalytic P Systems

We first consider purely catalytic P systems, which correspond to PPT systems where all sets of non-cooperative rules in the unique team work in the sequential derivation mode.

Theorem 5. *Every purely catalytic P system with n catalysts can be simulated by a corresponding globally homogenous PPT system of type ncoo, degree 1, and size n without applicability conditions working in the derivation mode sequ.*

Proof. The purely catalytic P system with n catalysts

$$\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, max),$$

i.e., $\mathcal{C} = \{c_k \mid 1 \leq k \leq n\}$, can be simulated by the globally homogenous PPT system of type *ncoo*, degree 1, and size n without applicability conditions working in the sequential derivation mode

$$\Pi = (V, \Sigma, P, T_1, sequ, w)$$

with $T_1 = \{R_{1,k} \mid 1 \leq k \leq n\}$ and

$$P = \{a \to u \mid c_k a \to c_k u \in \mathcal{R} \text{ for some } 1 \leq \text{k} \leq \text{n}\}$$

as well as

$$R_{1,k} = \{a \to u \mid c_k a \to c_k u \in \mathcal{R}\},\ 1 \leq k \leq n.$$

The applicability of the unique team works by applying (at most) one rule $a \to u$ from each $R_{1,k}$, $1 \leq k \leq n$, which corresponds to applying the corresponding rule $c_k a \to c_k u$ in Π. □

Simple catalytic P systems with n catalysts can be mimicked by simple P systems with prescribed teams of sets of rules, where as in the case of purely catalytic P systems the work of the n catalysts is simulated by n sets of non-cooperative rules in the team working in the sequential mode and one additional set of non-cooperative rules simulates the set of non-catalytic rules with working in the maximally parallel derivation mode.

Theorem 6. *Every catalytic P system with n catalysts can be simulated by a corresponding PPT system of type ncoo, degree 1, and size $n+1$ without applicability conditions with n components of the unique team working in the derivation mode sequ and one working in the derivation mode max.*

Proof. The catalytic P system with n catalysts

$$\Pi = (V, \mathcal{C}, \Sigma, w, \mathcal{R}, max),$$

i.e., $\mathcal{C} = \{c_k \mid 1 \leq k \leq n\}$, can be simulated by the PPT system of type *ncoo*, degree 1, and size $n+1$ without applicability conditions

$$\Pi = (V, \Sigma, P, T_1, w)$$

with $T_1 = \{R_{1,k} \mid 1 \leq k \leq n+1\}$ and

$$P = \{a \to u \mid c_k a \to c_k u \in \mathcal{R} \text{ for some } 1 \leq \text{k} \leq \text{n}\} \cup \{\text{a} \to \text{u} \mid \text{a} \to \text{u} \in \mathcal{R}\}$$

as well as

$$R_{1,k} = (\{a \to u \mid c_k a \to c_k u \in \mathcal{R}\}, sequ),\ 1 \leq k \leq n,$$

and

$$R_{1,n+1} = (\{a \to u \mid a \to u \in \mathcal{R}\}, max),$$

The application of the unique team works by applying (at most) one rule $a \to u$ from each $R_{1,k}$, $1 \leq k \leq n$, which corresponds to applying the corresponding rule $c_k a \to c_k u$ in Π, as well as the rules in $R_{1,n+1}$ in the maximally parallel way.

Observe that in contrast to the globally homogenous PPT systems for simulating purely catalytic P systems, we now have non-homogenous PPT systems, as we have to use both derivation modes *sequ* and *max* in the unique team. □

According to Propositions 2 and 1, from Theorems 5 and 6 we immediately infer the following results:

Corollary 1. *For any $d \geq 1$, we have*

1. $N^d\mathcal{L}(RE) = \mathcal{L}(PPT_{gen}(ncoo; 1, 3, *))$ *and*
2. $N^d\mathcal{L}(RE) = \mathcal{L}(PPT_{acc}(ncoo; 1, d+3, *))$*;*

moreover,
$Ps\mathcal{L}(RE) = \mathcal{L}(PPT_{gen}(ncoo; 1, *, *)) = \mathcal{L}(PPT_{acc}(ncoo; 1, *, *))$.
In all cases, the degree of the PPT systems is only 1.

8 PPT Systems Directly Simulating Register Machines

In this section we show how register machines can directly be simulated by PPT systems in an easy way when using applicability conditions represented by sets of permitting and forbidden contexts, see the defintion on page 9.

Theorem 7. *The computations of a register machine can be simulated by a globally homogenous PPT system of type ncoo and size* 2 *using permitting and forbidden contexts as applicability conditions in the sequential derivation mode.*

Proof. Consider the register machine

$$M = (m, B, l_0, l_h, R)$$

with B_{ADD} denoting the set of labels of *ADD*-instructions $p : (ADD(r), q, s)$ of arbitrary registers r, and $B_{SUB(r)}$ denoting the set of labels of all *SUB*-instructions $p : (SUB(r), q, s)$ of a decrementable register r. Moreover, for any $p \in B \setminus \{l_h\}$, $Reg(p)$ denotes the register affected by the *ADD*- or *SUB*-instruction labeled by p.

We now construct the globally homogenous PPT system of size 2 working in the sequential derivation mode using permitting and forbidden contexts as applicability conditions

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, sequ, w).$$

Throughout the computation of Π, one symbol $p \in B$ represents the instruction from the register machine to be simulated next, and the number of symbols a_r

represents the contents of register r, $1 \leq r \leq m$. Hence, we start with the axiom $w = l_0 w_0$, where w_0 represents the initial contents of the registers. If the final label l_h appears, we know that the computation in M has been successful and finally can erase l_h, so that a multiset over Σ remains as the result of the computation.

Moreover, the set of symbols V only consists of the labels in B and the symbols a_r representing the registers:

$$V = B \cup \{a_r \mid 1 \leq r \leq m\}.$$

The set of terminal symbols Σ consists of only those symbols from the set $\{a_r \mid 1 \leq r \leq m\}$ which represent output registers.

We need the following simple non-cooperative rules in P for the simulation of the instructions of M:

$$\begin{aligned} P = &\{p \to qa_r,\ p \to sa_r \mid p : (ADD(r), q, s) \in R\} \\ &\cup \{p \to q,\ p \to s \mid p : (SUB(r), q, s) \in R\} \\ &\cup \{a_r \to \lambda \mid 1 \leq r \leq m \text{ and r is a decrementable register}\} \\ &\cup \{l_h \to \lambda\} \end{aligned}$$

The teams of sets of rules with applicability conditions for simulating the instructions of the register machine defined below form the teams $T_1, \dots, T_n$.

- $p : (ADD(r), q, s)$, $p \in B_{ADD}$, is simulated by the team

  $R_p = \{((\{p\}, \emptyset), \{p \to qa_r,\ p \to sa_r\})\}$
  which can also be written in a simpler way as
  $R_p = \{\{p \to qa_r, p \to sa_r\}\}$, because the rules in this set of rules in this team can anyway only be applied if p is present.

- $p : (SUB(r), q, s)$; $p \in B_{SUB(r)}$, is simulated by the team of sets of rules

  $R_{p,1} = ((\{p, a_r\}, \emptyset), \{\{p \to q\}, \{a_r \to \lambda\}\})$

  (both the presence of p and a_r have to be checked in order to guarantee that both rules $p \to q$ and $a_r \to \lambda$ are applied) as well as by the team of sets of rules

  $R_{p,2} = \{((\{p\}, \{a_r\}), \{p \to s\})\}$

  (both the presence of p and the absence of a_r have to be checked to guarantee that we only proceed to label s if no symbol a_r is present).

- $l_h : HALT$ is simulated by the team
  $R_h = \{((\{l_h\}, \emptyset), \{l_h \to \lambda\})\}$,
  which can also be written in a simpler way as
  $R_h = \{\{l_h \to \lambda\}\}$.

Only in the case of applying a team $R_{p,1}$ two rules are applied in one step, otherwise only one rule is applied.

The application of a team is only possible if the current label symbol p appears in the underlying configuration, and in the case of a *SUB*-instruction also the presence/absence of $a_{Reg(p)}$ is correctly given. Throughout the computation in Π exactly one of the teams of sets of rules is applicable before finally a configuration only containing terminal symbols is reached. □

9 Computational Completeness

According to Subsection 2.1, register machines are a model being computationally complete for multisets. Hence, from Theorem 7 we immediately infer the following result:

Theorem 8. *PPT systems of type ncoo and size* 2 *when using applicability conditions represented by sets of permitting and forbidden contexts in the sequential derivation mode are computationally complete for multisets.*

Instead of non-cooperative rules we can also use the simple rules of insertion and deletion:

- $I(a)$ inserts an object a in the underlying multiset (and can be interpreted as the rule $\lambda \to a$).
- $D(a)$ deletes an object a from the underlying multiset, if at least one a is present (and can be interpreted as the rule $a \to \lambda$).

Based on the proof of Theorem 7, we easily get the following result for PPT systems using insertion and deletion rules (called PPT systems of type $InsDel$ for short):

Corollary 2. *PPT systems of type $InsDel$ and size* 3 *when using applicability conditions represented by sets of permitting and forbidden contexts in the sequential derivation mode are computationally complete for multisets.*

Proof. Consider the register machine

$$M = (m, B, l_0, l_h, R)$$

with B_{ADD} denoting the set of labels of *ADD*-instructions $p : (ADD(r), q, s)$ of arbitrary registers r, and $B_{SUB(r)}$ denoting the set of labels of all *SUB*-instructions $p : (SUB(r), q, s)$ of a decrementable register r. Moreover, for any $p \in B \setminus \{l_h\}$, $Reg(p)$ denotes the register affected by the *ADD*- or *SUB*-instruction labeled by p.

We now construct the globally homogenous PPT system of type $InsDel$ and size 3 working in the sequential derivation mode using permitting and forbidden contexts as applicability conditions

$$\Pi = (V, \Sigma, P, T_1, \ldots, T_n, sequ, w).$$

Throughout the computation of Π, one symbol $p \in B$ represents the instruction from the register machine to be simulated next, and the number of symbols a_r represents the contents of register r, $1 \leq r \leq m$. Hence, we start with the axiom $w = l_0 w_0$, where w_0 represents the initial contents of the registers. If the final label l_h appears, we know that the computation in M has been successful and finally can erase l_h, so that a multiset over Σ remains as the result of the computation.

Moreover, the set of symbols V only consists of the labels in B and the symbols a_r representing the registers:

$$V = B \cup \{a_r \mid 1 \leq r \leq m\}.$$

The set of terminal symbols Σ consists of only those symbols from the set $\{a_r \mid 1 \leq r \leq m\}$ which represent output registers.

We need the following simple insertion and deletion rules in P for the simulation of the instructions of M:

$$\begin{aligned} P = &\{I(p), D(p) \mid p \in B\} \\ &\cup \{I(a_r) \mid 1 \leq r \leq m\} \\ &\cup \{D(a_r) \mid 1 \leq r \leq m \text{ and r is a decrementable register}\} \end{aligned}$$

The teams of sets of rules with applicability conditions for simulating the instructions of the register machine defined below form the teams $T_1, \dots, T_n$.

- $p : (ADD(r), q, s)$, $p \in B_{ADD}$, is simulated by the team

  $$\begin{aligned} R_p = \{&((\{p\}, \emptyset), \{D(p)\}), \\ &((\{p\}, \emptyset), \{I(q),\ I(s)\}), \\ &((\{p\}, \emptyset), \{I(a_r)\})\}, \end{aligned}$$

  which in a shorter way could be written as

  $$R_p = ((\{p\}, \emptyset), \{\{D(p)\}, \{I(q),\ I(s)\}, \{I(a_r)\}\})$$

  as the applicability condition $(\{p\}, \emptyset)$ is required for all sets of rules in the team. Observe that the size of these teams now is 3!

- $p : (SUB(r), q, s)$; $p \in B_{SUB(r)}$, is simulated by the team

  $$R_{p,1} = ((\{p, a_r\}, \emptyset), \{\{D(p)\}, \{I(q)\}, \{D(a_r)\}\})$$

  (again the size of these teams now is 3)

  as well as by the team

  $$R_{p,2} = ((\{p\}, \{a_r\}), \{\{D(p)\}, \{I(s)\}\})$$

- $l_h : HALT$ is simulated by the team

$$R_h = \{\{D(l_h)\}\}.$$

Throughout the computation in Π exactly one of the teams of sets of rules is applicable before finally a configuration only containing terminal symbols is reached. □

As is well-known, catalytic and purely catalytic P systems are computationally complete (for multisets), too. Therefore, based on the results shown in Section 7 we get the following results (compare with Corollary 1):

Corollary 3. *Globally homogenous PPT systems of type ncoo and degree* 1 *without applicability conditions working in the sequential derivation mode are computationally complete for multisets.*

Corollary 4. *PPT systems of type ncoo and degree* 1 *without applicability conditions with one set of rules in the unique team working in the maximally parallel derivation mode and all the other sets of rules working in the sequential derivation mode are computationally complete for multisets.*

10 Conclusion

In this paper we have considered the concept of using prescribed teams of sets of rules being applied in different derivation modes, with the applicability of a team possibly depending on a given condition. Among other general results, we have shown that simple purely catalytic P systems with n catalysts can be simulated by simple P systems with one prescribed team of sets of rules with all n sets of non-cooperative rules in this team working in the sequential derivation mode thus simulating the work of the n catalysts, as well as that simple catalytic P systems with n catalysts can be simulated by simple P systems with one prescribed team of sets of non-cooperative rules, where one set of this team works in the maximally parallel derivation mode and the other n sets of rules in this team work in the sequential mode thus again simulating the work of the n catalysts. From the results known for simple (purely) catalytic P systems, we immediately infer the corresponding computational completeness results for the new variants of simple P systems, with on one hand only using non-cooperative rules and no applicability conditions. On the other hand, we can show computational completeness for different variants of simple P systems with prescribed teams of sets of non-cooperative rules by directly simulating register machines, thereby using applicability conditions given as sets of (atomic) promoters and inhibitors.

Throughout this paper, we have restricted ourselves to the two basic derivation modes, i.e., the sequential one and the maximally parallel derivation mode. A thorough investigation of simple P systems with prescribed teams of sets of rules using other derivation modes remains for future research. Moreover, other kinds of rules might be used, too; for example, insertion and deletion rules instead of non-cooperative rules as already used for simulating register machines in Corollary 2.

Acknowledgements

The authors would like to thank the Sevillan team for again organizing BWMC 2023, the 19th Brainstorming Week on Membrane Computing[4] this year and making it a fruitful event, also allowing for participation online. Artiom Alhazov acknowledges project 20.80009.5007.22 "Intelligent information systems for solving ill-structured problems, processing knowledge and big data" by the National Agency for Research and Development.

References

1. Alhazov, A., Freund, R., Ivanov, S.: When catalytic P systems with one catalyst can be computationally complete. Journal of Membrane Computing **3**(3), 170–181 (2021). https://doi.org/10.1007/s41965-021-00079-x
2. Alhazov, A., Freund, R., Ivanov, S., Oswald, M.: Variants of simple purely catalytic P systems with two catalysts. In: Vaszil, Gy., Zandron, C., Zhang, G. (eds.) International Conference on Membrane Computing ICMC 2021, Proceedings. pp. 39–53 (2021)
3. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: Variants of simple P systems with one catalyst being computationally complete. In: Vaszil, Gy., Zandron, C., Zhang, G. (eds.) International Conference on Membrane Computing ICMC 2021, Proceedings. pp. 21–38 (2021)
4. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: Prescribed teams of rules working on several objects. In: Durand-Lose, J., Vaszil, Gy. (eds.) Machines, Computations, and Universality – 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 – September 2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13419, pp. 27–41. Springer (2022). https://doi.org/10.1007/978-3-031-13502-6_6
5. Alhazov, A., Freund, R., Verlan, S.: P systems working in maximal variants of the set derivation mode. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10105, pp. 83–102. Springer (2017). https://doi.org/10.1007/978-3-319-54072-6_6
6. Csuhaj-Varjú, E., Dassow, J., Kelemen, J.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Topics in computer mathematics, Gordon and Breach (1994)
7. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer (1989)

[4] `http://www.gcn.us.es/19bwmc`

8. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 8340, pp. 173–188. Springer (2014). https://doi.org/10.1007/978-3-642-54239-8_13
9. Freund, R., Verlan, S.: A formal framework for static (tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 4860, pp. 271–284. Springer (2007). https://doi.org/10.1007/978-3-540-77312-2_17
10. Krithivasan, K., Păun, Gh., Ramanujan, A.: On controlled P systems. Fundam. Inform. **131**(3–4), 451–464 (2014). https://doi.org/10.3233/FI-2014-1025
11. Minsky, M.L.: Computation. Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ (1967)
12. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693
13. Păun, Gh.: Membrane Computing: An Introduction. Springer (2002). https://doi.org/10.1007/978-3-642-56196-2
14. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
15. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer (1997). https://doi.org/10.1007/978-3-642-59136-5
16. The P Systems Website. `http://ppage.psystems.eu/`

P Systems with Reactive Membranes

Artiom Alhazov[1], Rudolf Freund[2], Sergiu Ivanov[3], David Orellana-Martín[4,5], Antonio Ramírez-de-Arellano[4,5], José Antonio Rodríguez Gallego[6]

[1]Vladimir Andrunachievici Institute of Mathematics and Computer Science,
The State University of Moldova, Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2]Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[3]IBISC Laboratory, Université Paris-Saclay, Univ Évry
91020, Évry-Courcouronnes, France
E-mail : sergiu.ivanov@univ-evry.fr

[4]Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {dorellana,aramirezdearellano}@us.es

[5]SCORE Laboratory, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

[6]Departmento de Construcciones Arquitectónicas I,
Universidad de Sevilla
Avda. Reina Mercedes 2, 41012 Sevilla, Spain
E-mail: jrodriguez14@us.es

Summary. Membranes are one of the key concepts in P systems and membrane computing, and a lot of research focuses on their properties and possible extensions: membrane division, membrane dissolution, mobile membranes, etc. In this work, we explore the possibility of using membranes for thinking about the emergence of milieu separations at the origins of life. We propose a new variant of P systems with reactive membranes, in which every symbol is initially surrounded by an elementary membrane, and in which membranes can non-deterministically merge and split, leading to the formation of bigger and more complicated membranes. We show that such non-deterministic splitting and merging does not seem to radically affect the computational power: P systems with reactive membranes and non-cooperative rules generate at least all semilinear languages, and cooperative rules allow for simulating partially blind register machines. We briefly discuss using P systems with reactive membranes for illustrating the emergence of autocatalytic cycles, but actual constructions are left for future work.

Keywords: origins of life, P systems, self-assembly, space and topology.

1 Introduction

Membrane computing is a multiset rewriting-based theoretical construct for natural computing, originally introduced by Gh. Păun in [23], and extensively studied ever since. The structure of a membrane system—or a P system—mimics that of a living cell: it is a hierarchical family of nested membranes, each carrying a multiset of abstract objects and multiset rewriting rules. The objects can be seen as formal representations of chemical species, and the rewriting rules capture the biochemical interactions these species may have.

Beyond the obvious abstraction arrow between biochemical species and formal objects, membrane computing parallels biological systems in another interesting way. In biology, centralization of functions is quite frequent (e.g., central nervous systems, specialized organs, etc.), but not fundamental. Only as a first example, simple organisms carry out many activities in a decentralized way, weakly orchestrated by interference between related processes. Take unicellular organisms: a computer scientist may be tempted to consider the genetic material as the program for the whole cell, but it is now known (e.g. [10]) that the relationship between the genotype and the phenotype—its manifestation—is very far from the clear program–execution duality imbuing computer science. As an abstraction of hierarchically structured biochemistry, P systems inherit this weakly centralized way of functioning, which makes them a good candidate for supporting the thought process about some grand laws of biology.

In this paper, we lay the groundwork for using P systems as a tool for thinking about some aspects of the emergence of life. The particular question we focus on is the emergence of milieu separations, which played an essential role as they allowed to isolate and protect relevant processes from the environment [11]. Since P systems already include membranes as first-class citizens, we will use them as a framework for thinking about the emergence of complex regions from simpler ones.

The approach we take here is to posit that every copy of a symbol a is endowed with some *elementary space*—a membrane which initially only contains the multiset a. Two such symbols can bond by merging their membranes, thereby yielding a more complex membrane containing 2 symbols. Such membranes can further merge, yielding bigger and bigger regions. Dually, membranes containing multiple symbols can split into a pair of simpler membranes, with the content of the original larger membrane distributed across its children. This is in fact membrane separation (e.g. [7, 21, 22]).

Measuring the complexity of a membrane by the number of symbols it contains is simultaneously simple and appropriate: cooperative evolution rules are allowed, so more symbols means more applicable rules and therefore more interactions. In the setup we establish in this paper, all membranes share the same common set of evolution rules. The rules can naturally be seen as defining a chemistry, while membrane merging and splitting can on the other hand be seen as some lower-level ground laws governing who may interact with whom, i.e. the topology of the interactions. The resulting abstract structures featuring merging and splitting membranes are therefore systems in which objects interact based on the non-deterministic variations in their neighborhoods. We call such structures P systems with *reactive membranes*.

Before using P systems with reactive membranes as a formal tool, a number of important details have to be sorted out. In particular, we show that the definition of membrane splitting and merging turns out rather nontrivial. Choosing when to recover and how to interpret the result also impacts the form of the computations of a P system with reactive membranes, and also what kind of results one can expect. Finally, this P system variant as informally introduced above and defined in Section 3 is very basic and may be extended in many ways, as we briefly show in Section 5.

Note that we do not pretend to faithfully model in any way the processes which happened at the origins of life. Rather, we acknowledge the exceptional complexity of these processes, as well as the impossibility to experimentally verify any of the related hypotheses (e.g., [17]). The intended role of P systems with reactive membranes is to serve as a formal vehicle for an otherwise abstract thought process, to help verify the latter in a basic way, and to help the researcher to deal with complex questions. This approach is similar in spirit to the works [26, 27], in which sign Boolean networks are used with a similar purpose.

P systems with reactive membranes are naturally part of the lineage of P systems with active membranes, and feature similarities with other variants in this family. Among closely related variants are P systems with mobile membranes, in which membranes are allowed to move across the membrane structure, and thereby change their immediate neighbors [8, 9, 20]. Another variant are P systems with vesicles of multisets, in which multisets are contained in vesicles, which are contained in membranes, implying that entire multisets of symbols can travel between different membranes, thereby activating different sets of rules [5, 15]. A key specificity of P systems with reactive membranes setting them apart from the other variants is that membrane splitting and merging is global, compulsory, and independent of the contents of the membranes or of the rules. This feature introduces a basic form of space, through which the entities travel and in which they interact in their immediate neighborhood. On the other hand, such compulsory splitting and merging modulates the computational power in interesting ways.

This paper is structured as follows. In Section 2 we recall some basic concepts from formal languages and P systems. In Section 3 we introduce P systems with reactive membranes, and define the precise semantics of splitting and merging of membranes. In Section 4 we present some first results concerning the computational power of P systems with reactive membranes, with non-cooperative and cooperative rules. In Section 5 we give some examples of possible extensions to the new variant. Finally, in Section 6, we discuss the potential of reactive membranes for illustrating some processes which happened at the origins of life, as well as some aspects of their computational power.

2 Preliminaries

For an alphabet V, a finite non-empty set of abstract symbols, the free monoid generated by V under the operation of concatenation, i.e., the set containing all possible strings over V, is denoted by V^*. The empty string is denoted by λ, and $V^* \backslash \{\lambda\}$ is denoted by V^+.

For two natural numbers $a, b \in \mathbb{N}$, $a \leq b$, we use the notation $[a..b]$ to refer to the interval of natural numbers between a and b, both included: $[a..b] = \{a, a+1, \ldots, b\}$.

Given a finite set A, a multiset over A is a function $w : A \to \mathbb{N}$, assigning the number of times an element of A appears in w. The infinite set of all multisets over A is denoted by A°. The family of finite sets of finite multisets over A is denoted by $\mathcal{P}_{fin}(A^\circ)$.

To spell out a multiset w, we will generally write any string containing exactly the same symbols with the same multiplicities. For example, the strings aab, aba, ba^2 will be used to refer to the same multiset w with the property $w(a) = 2$, $w(b) = 1$, and $w(c) = 0$ for all $c \in A \setminus \{a, b\}$. We denote the empty multiset by Λ, i.e. $\forall a \in A : \Lambda(a) = 0$, and its string representation is λ, the empty string.

Given two multisets w_1 and w_2 over A, their multiset union $w_1 \cup w_2$ is defined as $(w_1 \cup w_2)(a) = w_1(a) + w_2(a)$, for all $a \in A$. As their multiset intersection $w_1 \cap w_2$ we define $(w_1 \cap w_2)(a) = \min\{w_1(a), w_2(a)\}$. A restriction of the multiset $w : A \to \mathbb{N}$ to the subset $B \subseteq A$ is the multiset $w|_B : A \to \mathbb{N}$ with the property that $w|_B(a) = w(a)$ if $a \in B$ and $w|_B(a) = 0$ otherwise.

The family of regular, context-free, and recursively enumerable string languages is denoted by $\mathcal{L}(REG)$, $\mathcal{L}(CF)$, and $\mathcal{L}(RE)$, respectively. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. As $Ps\mathcal{L}(REG) = Ps\mathcal{L}(CF)$, in the area of multiset rewriting $\mathcal{L}(CF)$ plays no role at all, and in the area of membrane computing we often only get characterizations of $Ps\mathcal{L}(REG)$ and $Ps\mathcal{L}(RE)$.

For further notions and results in formal language theory we refer to textbooks like [12] and [25].

In the rest of this section, we briefly recall P systems and the related concepts. For more extensive overviews, we refer the reader to [18, 24].

A *(transition) P system* is a construct

$$\Pi = (O, T, \mu, w_1, \ldots, w_n, R_1, \ldots R_n, \delta, h_i, h_o) \text{ where}$$

- O is the alphabet of *objects*,
- $T \subseteq O$ is the alphabet of *terminal objects*,
- μ is the *membrane structure* injectivley labelled by the numbers from $\{1, \ldots, n\}$ and usually given by a sequence of correctly nested brackets,
- w_i are the multisets giving the *initial contents* of each membrane i, $1 \leq i \leq n$,
- R_i is the finite *set of rules* associated with membrane i, $1 \leq i \leq n$,
- δ is the *derivation mode*, and

- h_i and h_o are the labels of the *input membrane* and the *output membrane*, respectively; $1 \leq h_i \leq n$, $1 \leq h_o \leq n$).

Taking into account the well-known flattening process, which means that computations in a P system with an arbitrary (static) membrane structure can be simulated in a P system with only one membrane, e.g., see [14], often only *simple P systems* are considered, i.e., with the simplest membrane structure of only one membrane region, and then we write:

$$\Pi = (O, T, w_1, R_1, \delta)$$

Quite often, the rules associated with membranes are multiset rewriting rules (or special cases of such rules). Multiset rewriting rules have the form $u \to v$, with $u \in O^\circ \setminus \{\lambda\}$ and $v \in O^\circ$, where O° is the set of multisets over O, and $\lambda(a) = 0$, for all $a \in O$. If $|u| = 1$, the rule $u \to v$ is called *non-cooperative*, otherwise it is called *cooperative*. In *communication P systems*, rules are additionally allowed to send symbols to the neighbouring membranes. In this case, for rules in R_i, $v \in (O \times Tar_i)^\circ$, where Tar_i contains the symbols *out* (corresponding to sending the symbol to the parent membrane), *here* (indicating that the symbol should be kept in membrane i), and in_j (indicating that the symbol should be sent into the child membrane j of membrane i). When writing out the multisets over $O \times Tar_i$, the indication *here* is often omitted.

In P systems, rules are often applied in the maximally parallel way: in one derivation step, only a non-extendable multiset of rules can be applied. The rules are not allowed to consume the same instance of a symbol twice, which creates competition for objects and may lead to non-deterministic choice between the maximal collections of rules applicable in one step. The *maximally parallel derivation mode* is generally denoted by the symbol *max*. Other derivation modes include the *sequential derivation mode sequ* in which exactly one rule is applied in every step, the *set maximally parallel derivation mode smax* only allowing multisets of rules in which every rule has multiplicity 1, as well as the *asynchronous derivation mode asyn* under which no restriction is imposed on the applied multiset of rules. We refer to the works [3, 4, 6, 16] for an in-depth discussion of the matter.

A *computation* of a P system is traditionally considered to be a sequence of configurations it can successively visit, stopping at the halting configuration. A *halting* configuration is a configuration in which no rule can be applied any more, in any membrane. The *result of a computation* in a P system Π as defined above is the contents of the output membrane h_o projected over the terminal alphabet T.

We will use the notations $N(\Pi)$ and $Ps(\Pi)$ to respectively refer to the number language and the language of multisets generated by Π. The notation $OP_n(\delta, \tau)$ will refer to the family of P systems with at most n membranes, operating under the derivation mode δ and relying on the rules of type τ, where $\tau = coo$ if cooperative rules are allowed and $\tau = ncoo$ if all rules are non-cooperative. Finally, we use the notations $NOP_n(\delta, \tau)$ and $PsOP_n(\delta, \tau)$ to refer to the family of number languages and multiset languages, respectively, generated by the P systems in the family $OP_n(\delta, \tau)$.

Example 1. Figure 1 shows the graphical representation of the P system formally given by

$$\begin{aligned} \Pi &= (\{a,b,c,d\},\{a,d\},[_1[_2]_2]_1,R_1,R_2,1,1), \\ R_2 &= \{a \to aa, b \to b\,(c, out)\}, \\ R_1 &= \emptyset. \end{aligned}$$

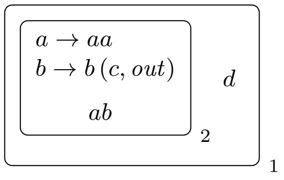

Fig. 1. An example of a simple P system.

In the maximally parallel mode, the inner membrane 2 of Π will apply as many instances of the rules as possible, thereby doubling the number of a, and ejecting a copy of c into the surrounding (skin) membrane in each step. The symbol d in the skin membrane is not used. Therefore, after k steps of evolution, membrane 2 will contain the multiset $a^{2^k}b$ and membrane 1 the multiset $c^k d$. Since all rules are always applicable in Π, this P system never halts. □

3 Reactive Membranes

A *P system with reactive membranes* is the following construct:

$$\Pi = (O, T, W_0, R, \delta) \quad \text{where}$$

- O is the alphabet of *objects*,
- $T \subseteq O$ is the alphabet of *terminal objects*,
- $W_0 \in \mathcal{P}_{fin}(O^\circ)$ is the (finite) *initial set of multisets* over O,
- $R \subseteq O^\circ \times O^\circ$ is the set of *evolution rules*, and
- δ is the *derivation mode.*

We will require that at least one of the sides of all rules in R be non-empty, i.e.

$$\forall u \to v \in R : u \neq \lambda \vee v \neq \lambda.$$

We immediately stress two major features of this definition. On the one hand, we do not include any membrane structure. Indeed, as W_0 hints, we simply use individual multisets to represent the contents of the individual membranes, without explicitly representing the membranes themselves. Incidentally, this means that membranes do not nest in this model. On the other hand, the evolution of all symbols in all multisets is governed by the same common set of rules R.

A configuration of Π is any set of multisets over O. Similarly to networked models of computing like networks of evolutionary processors (e.g. [19]) or tissue P systems with vesicles of multisets [5], a computation step in P systems with reactive membranes consists of two stages:

1. splitting and merging,
2. evolution.

Informally, the splitting and merging stage implements the non-deterministic evolution of the membranes—individual multisets under this definition—as described in the introduction: any two multisets may merge, and any multiset may split in two. The evolution stage consists in applying the evolution rules in R to every multiset of the configuration, according to the mode δ. In the following paragraphs we give a formal description of both stages, applied to a configuration $W_i \in \mathcal{P}_{\mathit{fin}}(O^\circ)$.

Splitting and merging stage

1. Non-deterministically partition W_i into 3 subsets:
$$W_i = M_i \cup S_i \cup I_i$$
such that $|M_i|$ is even, and the sets S_i, M_i, and I_i are mutually disjoint, i.e., $S_i \cap M_i = S_i \cap I_i = M_i \cap I_i = \emptyset$. The multisets in M_i will be merged pairwise, the multisets in S_i will be split, and the multisets in I_i will remain intact.
2. Partition M_i into a set of disjoint pairs. Non-deterministically pick a bijection $\varphi : [1..|M_i|] \to M_i$ and construct the following set:
$$\hat{M}_i = \{(\varphi(2k-1), \varphi(2k)) \mid 1 \le k \le |M_i|/2\}.$$
Then define $M_i' = \{w_1 \cup w_2 \mid (w_1, w_2) \in \hat{M}_i\}$.
3. Define split(w) to be the set of all possible ways to split the multiset w into two multisets:
$$\text{split}(w) = \{(w_1, w_2) \mid w_1 \cup w_2 = w, w_1, w_2 \in O^\circ\}.$$
Define the set of all possible ways of splitting the multisets in S_i:
$$\hat{S}_i = \prod_{w \in S_i} \text{split}(w).$$
Non-deterministically pick $S_i' \in \hat{S}_i$.
4. Compute the new intermediate configuration as
$$W_i' = M_i' \cup \text{flatten}(S_i') \cup I_i,$$
where $\text{flatten}(S_i') = \{w_1, w_2 \mid (w_1, w_2) \in S_i'\}$.

In the above presentation we describe merging before splitting, but the order of the two substeps does not matter, since they occur on disjoint sets M_i and S_i. Furthermore, we stress that multiple intermediate configurations W_i' may be obtained from the same configuration W_i.

Evolution stage

The evolution stage is defined in the conventional way by applying the rules in R to every multiset in W_i' individually, according to the derivation mode δ:

$$W_i = \{w' \in W_i' \mid w \xRightarrow{\delta,R} w'\},$$

where w' is a multiset derived from w by applying the rules in R under the mode δ.

A configuration W is *halting* if no rules are applicable in the evolution stage, for any intermediate configuration W' which can be obtained from W in the splitting and merging stage. An *n-step halting computation* of a P system with reactive membranes Π is a finite sequence of configurations $(W_i)_{0\leq i\leq n}$ such that W_{i+1} is obtained from W_i by the computation step described above, and W_n is a halting configuration.

As the *result of a computation* in a P system with reactive membranes Π as defined above we take all the terminal objects appearing in the membranes present in a halting configuration W_n:

$$\left(\bigcup_{w\in W_n} w\right)\Bigg|_T = \bigcup_{w\in W_n} w|_T .$$

To conclude the introduction of P systems with reactive membranes, we again stress that the splitting and merging of multisets (or membranes) is non-deterministic, imposed in every computation step, and independent of the features of the configuration or of the rules in R. More concretely, the rules in R cannot directly influence which symbols will appear next to which after the splitting and merging stage.

Example 2. Consider the following P system with reactive membranes:

$$\begin{aligned}
\Pi &= (O, T, W_0, R, max), \text{ where}\\
O &= \{a, b, c, d, e, f\},\\
T &= \{d, f\},\\
W_0 &= \{a, b, c\},\\
R &= \{ab \to d, abc \to f, a \to e\}.
\end{aligned}$$

For the first step of the computation, Π may decide to not split or merge any multisets ($M_0 = S_0 = \emptyset$, $I_0 = W_0$), meaning that the evolution rules will be applied directly to singleton multisets a, b, and c. While no rules are applicable to b or c individually, the rule $a \to e$ will have to be applied to a, yielding the next configuration $W_1 = \{b, c, e\}$. We can immediately conclude that the rules $ab \to d$ and $abc \to f$ will never be applicable any more later in this computation, as there is no way to reintroduce a. In sum, this halting computation yields the result Λ.

Suppose now that Π decides to merge the multisets a and b in the first step, yielding the intermediate configuration $W_0' = \{ab, c\}$. In this case a non-deterministic choice will appear in the evolution stage between applying the rule $ab \to d$ or $a \to e$ (both singleton sets of rules are non-extendable). As a consequence, the following two possibilities exist for the second configuration: $\{d, c\}$ and $\{eb, c\}$. In sum, these halting computations yield the results d and Λ, respectively.

Finally, note that the rule $abc \to f$ will never be applicable with $W_0 = \{a, b, c\}$, since putting a, b, and c together in one membrane requires at least two mergers, and a will necessarily be consumed by $a \to e$ or $ab \to d$ along the way. On the other hand, if we put together a, b and c in one multiset from the start, or even if we put ab together and c apart, the rule $abc \to f$ will have a chance to be applied. In particular, in the case in which the initial configuration is $\{ab, c\}$ it suffices to consider the branch of the computation along which Π decides to merge the two multisets in the first splitting and merging stage. In sum, with the initial sets $\{ab, c\}$ and $\{a, b, c\}$ we can get the results Λ, d, and f. □

As indicated by the example discussed above, it makes a difference in how many multisets the initial multiset of objects is divided. Thus, we will use the notation $Re_nOP(\delta, \tau)$ to refer to the family of P systems with reactive membranes starting with n initial multisets, running under the mode δ and using rules of type $\tau \in \{coo, ncoo\}$, as well as the notations $NRe_nOP(\delta, \tau)$ and $PsRe_nOP(\delta, \tau)$ to refer to the family of number languages and multiset languages, respectively, generated by the P systems with reactive membranes from $Re_nOP(\delta, \tau)$.

Whereas on the one hand the previous example shows the effect of having more than one initial membrane, prohibiting the application of some evolution rules, the next example shows that the halting condition can be fulfilled due to the fact that symbols are distributed over several membranes, although some rule could be applied if all symbols on its left-hand side could be put into the same membrane by a merge operation. As merging can only combine the contents of two membranes, we can already get the situation that a rule with three symbols in its left-hand side cannot be applied any more.

Example 3. Consider the following P system with reactive membranes:

$$\begin{aligned}
\Pi &= (O, T, W_0, R, max), \text{ where} \\
O &= \{a_i, a_i', a_i'' \mid 1 \le i \le 3\} \cup \{f\}, \\
T &= \{a_1'', a_2'', a_3'', f\}, \\
W_0 &= \{a_1a_2a_3\}, \\
R &= \{a_i \to a_i', a_i' \to a_i'', a_1''a_2''a_3'' \to f\}.
\end{aligned}$$

If in the first two steps of the computation, Π decides to not split or merge any multisets, from $W_0 = \{a_1a_2a_3\}$ with applying the rules $\{a_i \to a_i' \mid 1 \le i \le 3\}$, after the first evolution step we obtain $W_1 = \{a_1'a_2'a_3'\}$, and by then applying the rules $\{a_i' \to a_i'' \mid 1 \le i \le 3\}$, after the second evolution step we obtain $W_2 = \{a_1''a_2''a_3''\}$. Keeping $\{a_1''a_2''a_3''\}$ in the same membrane then allows for applying the rule $a''_1a''_2a''_3 \to f$, thus obtaining the terminal result $W_3 = \{f\}$, as W_3 is a halting configuration.

Yet with two splits, but still applying the rules $\{a_i \to a'_i \mid 1 \leq i \leq 3\}$ in the first evolution step and the rules $\{a'_i \to a''_i \mid 1 \leq i \leq 3\}$ in the second evolution step, we get a two-step halting computation

$$\{a_1a_2a_3\} \Longrightarrow \{a'_1, a'_2a'_3\} \Longrightarrow \{a''_1, a''_2, a''_3\}$$

yielding the terminal result $a''_1a''_2a''_3$.

We also mention that with having $T = \{f\}$ only, this halting computation yields the result Λ. □

4 Computational Power: First Results

In this section, we list some first results regarding the computational power of P systems with reactive membranes. We start by remarking that the halting condition can checked in an easier way when the system only includes non-cooperative rules.

Remark 1. When using only non-cooperative rules, the halting condition for a configuration W can be checked without considering all possible splits and mergers and then the non-applicability of the rules in all membranes; instead it suffices to check the non-applicability of the rules to the flatten(W), i.e., to the union of multisets in all the membranes of W.

The following result even holds for non-cooperative rules and cooperative rules.

Lemma 1. *For every $Re_1OP(\delta, \tau)$ system there exist an equivalent $Re_nOP(\delta, \tau)$ system, for every $n > 1$.*

Proof. Given a P system with reactive membranes using rules of type τ

$$\Pi' = (O, T, \{w\}, R, \delta),$$

an equivalent P system with reactive membranes using rules of type τ with n initial membranes is

$$\Pi' = (O, T, \{w, w_2 = \Lambda, \dots, w_n = \Lambda\}, R, \delta).$$

In an empty membrane Λ, no non-cooperative rules or cooperative rules are applicable. Moreover, merging a membrane X with Λ yields X again, so no additional applications of rules can happen.

Now we show that splitting and merging do not affect the (results of the) computations in a P system with reactive membranes at all, no matter which derivation mode is used, when only non-cooperative rules are used. Hence, we get a characterization of $Ps\mathcal{L}(REG)$:

Theorem 1. *For any $\delta_1, \delta_2 \in \{asyn, seq, max, smax\}$ and $Y \in \{N, Ps\}$ as well as any $n \geq 1$,*

$$YRe_nOP(\delta_1, ncoo) = YOP_1(\delta_2, ncoo) = Y\mathcal{L}(REG).$$

Proof. The equality $YOP_1(\delta_2, ncoo) = Y\mathcal{L}(REG)$ is folklore, e.g., see [24]. The main idea for proving this result is that the evolution of symbols by applying non-cooperative rules can be described by a derivation tree, but for the resulting terminal objects it is completely irrelevant when the symbols evolve.

A similar argument now can be used here to argue that for any $\delta_1 \in \{asyn, seq, max, smax\}$,

$$YRe_nOP(\delta_1, ncoo) = YOP_1(asyn, ncoo) = Y\mathcal{L}(REG).$$

($\Rightarrow$) Given a P system with reactive membranes

$$\Pi' = (O, T, \{w\}, R, \delta_1),$$

we can easily define the equivalent simple P system

$$\Pi = (O, T, w, R, asyn).$$

Even distributing the contents of a single membrane over several membranes, even at the beginning with having several initial multisets, does not effect the applicability of the non-cooperative rules. Yet we have to mention that using the sequential derivation mode in several membranes yields a kind of parallelism like *smax*, but also this has no effect on the results of computations, especially as, according to Remark 1, halting only depends on the non-applicabiltiy of all rules to the symbols in all the multisets of the underlying configuration.

($\Leftarrow$) Given a simple P system

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

we can easily define the equivalent P system with reactive membranes

$$\Pi' = (O, T, \{w\}, R, asyn).$$

Any derivation of the 1-membrane transition P system Π operating under the the asynchronous derivation mode can be directly simulated by the P system with reactive membranes Π' which uses the same rules for the evolution stage, but then always chooses to not split or merge any membranes, i.e. M_i and S_i from the splitting and merging stage are always empty. As we are only using non-cooperative rules, the applicability of all the (multisets of) rules applied in Π is also guaranteed in Π'.
Finally we can apply Lemma 1 to get an equivalent P system with reactive membranes with n initial membranes.

In sum we see that P systems with reactive membranes behave as the corresponding transition P system when only non-cooperative rules are used. □

Finally, P systems with reactive membranes working under the maximally parallel mode and using cooperative rules can simulate partially blind register machines. As a reminder, we mention that partially blind register machines (PBRM) have programs consisting of the following two types of instructions for incrementing and decrementing a register:

- $(p, \mathrm{ADD}(r), q, s)$: in state p increment register r and jump to state q or state s;
- $(p, \mathrm{SUB}(r), q)$: in state p try to decrement register r; if successful, jump to state q, otherwise abort the computation without producing a result.

Partially blind register machines feature a final zero check: the register machine only halts with producing a result if all non-output registers are empty when the machine reaches the halting instruction uniquely labeled by h.

We will refer to the set of multiset languages generated by partially blind register machines by $PsPBRM$.

Theorem 2. *For any $\delta \in \{asyn, sequ, max, smax\}$,*

$$PsPBRM \subseteq PsRe_1OP(\delta, coo).$$

Proof (Sketch). The main idea of the proof is that throughout the simulation of the partially blind register machine, the configurations of the P system with reactive membranes Π always contains exactly one instance of the symbol representing the label of the instruction to be carried out next. The contents of a register r is represented by the total number of symbols a_r in the configurations of Π.

The increment instruction $(p, \mathrm{ADD}(r), q, s)$ can be simulated directly by the rules $p \to qa_r$ and $p \to sa_r$.

The decrement instruction $(p, \mathrm{SUB}(r), q)$ can be simulated by the following two rules: $pa_r \to q$, $p \to p$. Moreover, for every register symbol a_r with r not being an output register, we add the unit rules $a_r \to a_r$.

Indeed, if p and a copy of a_r find themselves in the same membrane, then a successful decrement is simulated: the total number of copies of a_r in the system is reduced by one.

If there are no copies of a_r left in the system, then p only has the chance to be used with the unit rule $p \to p$; observe that in any derivation mode at least one rule has to be applied if the system is not halting, i.e., as long as there still is a rule which can be applied to some symbol. In this case, either $p \to p$ and/or some unit rule $a_r \to a_r$ can be applied in every future derivation step, hence, the computation will never halt.

If copies of a_r do appear in the system, but not in the membrane containing p, then p can use the unit rule $p \to p$, and in any derivation mode either only this rule and/or other unit rules $a_r \to a_r$ can be applied. If in some future step, p and a_r appear in the same membrane, possibly $pa_r \to q$ can be applied. Otherwise, again we obtain just non-halting computation branches.

However, there must exist another branch in which no splits and mergers have happened at all, i.e., p and a_r are together, and in which the simulation therefore will be able to proceed correctly. The same alternative holds if p and a_r share the same membrane, but the system non-deterministically would choose to only apply $p \to p$ rather than $pa_r \to p$.

As soon as the halting label h appears, we have to use the final rule $h \to \lambda$. The final zero check is simulated by the unit rules $a_r \to a_r$ for all non-output registers r, which keep the computation to go on forever if at least one such symbol a_r is still present. Observe that this argument does not depend on the distribution of the symbols in the membranes of a configuration.

In sum, we conclude that the P system with reactive membranes Π can simulate the computations of the given partially blind register machine correctly, but on the other hand cannot yield more results. □

Finally, we remark that the construction we show here is non-deterministic, even if the simulated partially blind register machine is deterministic, i.e., all increment instructions are of the form $(p, \mathrm{ADD}(r), q, q)$, which in a simpler way can be written as $(p, \mathrm{ADD}(r), q)$.

5 Extensions

Given the motivation to use P systems with reactive membranes for thinking about the emergence of space and space separations in abiotic environments, and also the richness of the ecosystem of P systems variants, multiple extensions can be proposed.

A natural one to be considered would be *limiting the size* of individual membranes, as real membranes do not generally grow very big. Limitations on the number of symbols have already been considered in P systems [2], but combined with constant splitting and merging this ingredient may have a drastically different impact. It would be necessary to decide what happens when a membrane attains its maximal capacity. The approach in [2] is to prevent it to accept new symbols, but in the context of reactive membranes it may be appropriate to bias the splitting and merging stage of the computational step to force such full membranes to split. The contribution of such limitations to the computational power is yet unclear, but probably in some strong relation to the size of the left-hand sides of the evolution rules.

An extension in the spirit of generalized P systems [13] would be to *subject the rules to splitting and merging*. With such an extension, membranes would contain objects and rules, and splitting and merging would affect not only which symbols can interact, but also which rules will ensure their interaction.

Finally, splitting and merging could also be applied to rules: for example, a rule $u \to v$ could split into two rules $u \to \alpha$ and $\alpha \to v$, which could later merge back into $u \to v$. Similarly to splitting and merging of membranes, splitting and merging of rules delays some interactions. Relevance to thinking about the origins of life and the computational power of this variant remain to be explored.

6 Conclusion and Perspectives

This paper is a first attempt at using P systems for thinking about the origins of life, and in particular about the emergence of individual compartments separated by membranes. We introduced P systems with reactive membranes, in which every symbol is conceptually surrounded by elementary membranes, which then can merge to form bigger membranes, or split. Mimicking biochemistry, the set of rules is common to all membranes—the differences in the processes in different membranes should come from the symbols. Cooperative rules are allowed, and probably even necessary to meaningfully implement distinctions between membranes.

It is still an open research direction to actually illustrate some processes believed to have happened during abiogenesis in P systems with reactive membranes. Perhaps the most promising would be to implement autocatalytic cycles (e.g. [11]). The next step would be to implement *self-replication*, as suggested by José M. Sempere in a discussion. Indeed, in P systems with reactive membranes the membrane structure emerges spontaneously, which makes them a promising candidate for implementing self-replication of something other than symbol objects.

A parallel research direction which we started to explore in this paper is the computational power of P systems with reactive membranes. We have shown here that splitting and merging does not affect the computational power of P systems with reactive membranes using non-cooperative rules—P systems with reactive membranes using non-cooperative rules have the same computational power as simple P systems provided we only start with one singleton multiset, no matter which derivation mode we use. Based on this result, we have shown that P systems with reactive membranes can characterize the family of Parikh sets of semilinear languages when using only non-cooperative rules in any derivation mode.

Finally, when cooperative rules are allowed, P systems with reactive membranes can generate all multiset languages generated by partially blind register machines.

Several questions still remain to be addressed, in particular: can splitting and merging *augment* the computational power? It would indeed be surprising, but it has already been shown that non-deterministic shuffling of rule right-hand sides allows for generating non-semilinear languages [1], meaning that random shuffling of symbol neighborhoods as described in this paper may boost the power of the variant in some specific cases.

A subtle aspect which we do not discuss in depth in this paper is the halting condition and the procedure for retrieving the result. There is an asymmetry between these two: halting occurs when no more evolution rules are applicable after all possible splits and mergers. On the other hand, getting out the result essentially happens by merging *all* membranes into a single one.

Since the computational results we give in this paper seem to depend directly on the halting condition and on the procedure for obtaining the result, it would be relevant to explore how slight variations in these two affect the computational power of P systems with reactive membranes.

Finally, in Section 5 we have suggested several possible extensions of the new variant. A formal exploration of the computational power of such extensions would be quite relevant. Even more importantly, it would be very relevant to identify which extensions are more useful for using P systems with reactive membranes in thinking about the origins of life.

Acknowledgements

The authors would like to thank the Organizing Committee of the 19th Brainstorming Week on Membrane Computing[1] (BWMC 2023) for organizing this fruitful event. The authors would also like to thank José M. Sempere for multiple helpful suggestions about self-replication and emergence of space.

Artiom Alhazov acknowledges project 20.80009.5007.22 “Intelligent information systems for solving ill-structured problems, processing knowledge and big data” by the National Agency for Research and Development.

References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. P systems with randomized right-hand sides of rules. *Theor. Comput. Sci.*, 805:144–160, 2020.
2. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. P systems with limited number of objects. *J. Membr. Comput.*, 3(1):1–9, 2021.
3. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Marion Oswald. Variants of derivation modes for which purely catalytic P systems are computationally complete. *Theor. Comput. Sci.*, 920:95–112, 2022.
4. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Sergey Verlan. Variants of derivation modes for which catalytic P systems with one catalyst are computationally complete. *J. Membr. Comput.*, 3(4):233–245, 2021.
5. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Sergey Verlan. Tissue P systems with vesicles of multisets. *Int. J. Found. Comput. Sci.*, 33(3&4):179–202, 2022.
6. Artiom Alhazov, Rudolf Freund, and Sergey Verlan. P systems working in maximal variants of the set derivation mode. In Alberto Leporati, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers*, volume 10105 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2016.
7. Artiom Alhazov and Tseren-Onolt Ishdorj. Membrane operations in P systems with active membranes. *Second Brainstorming Week on Membrane Computing*, pages 37–44, 2004.
8. Bogdan Aman and Gabriel Ciobanu. Simple, enhanced and mutual mobile membranes. *Trans. Comp. Sys. Biology*, 11:26–44, 2009.

[1] `http://www.gcn.us.es/19bwmc`

9. Luca Cardelli and Gheorghe Păun. An universality result for a (mem)brane calculus based on mate/drip operations. *Int. J. Found. Comput. Sci.*, 17(1):49–68, 2006.
10. Matthew Cobb. 60 years ago, Francis Crick changed the logic of biology. *PLOS Biology*, 15(9):1–8, 09 2017.
11. Bruce Damer and David Deamer. The hot spring hypothesis for an origin of life. *Astrobiology*, 20(4):429–452, 2020.
12. Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
13. Rudolf Freund. Generalized P-systems. In Gabriel Ciobanu and Gheorghe Păun, editors, *Fundamentals of Computation Theory, 12th International Symposium, FCT '99, Iasi, Romania, August 30 - September 3, 1999, Proceedings*, volume 1684 of *Lecture Notes in Computer Science*, pages 281–292. Springer, 1999.
14. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Claudio Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.
15. Rudolf Freund and Marion Oswald. Tissue P systems and (mem)brane systems with mate and drip operations working on strings. In Nadia Busi and Claudio Zandron, editors, *Proceedings of the First Workshop on Membrane Computing and Biologically Inspired Process Calculi, MeCBIC@ICALP 2006, Venice, Italy, July 9, 2006*, volume 171 (2) of *Electronic Notes in Theoretical Computer Science*, pages 105–115. Elsevier, 2006.
16. Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) P systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2007.
17. Nicolas Glade. *Le Vivant Rare, Faible et Amorphe - Évolution depuis les Origines jusqu'à la Vie telle qu'elle nous Apparaît. (Rare, Weak and Amorphous Life - Evolution of Life from the Origins until Life as it Appears Nowadays)*. HAL Open Archive, 2022.
18. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
19. Sergiu Ivanov, Yurii Rogozhin, and Sergey Verlan. Small universal networks of evolutionary processors. *J. Autom. Lang. Comb.*, 19(1-4):133–144, 2014.
20. Shankara Narayanan Krishna and Gheorghe Păun. P systems with mobile membranes. *Nat. Comput.*, 4(3):255–274, 2005.
21. David Orellana-Martín, Luis Valencia-Cabrera, and Mario J. Pérez-Jiménez. P systems with evolutional communication and separation rules. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality – 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 – September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 2022.
22. Linqiang Pan and Tseren-Onolt Ishdorj. P systems with active membranes and separation rules. *J. Univers. Comput. Sci.*, 10(5):630–649, 2004.
23. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

24. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
25. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages.* Springer, 1997.
26. Rémi Segretain, Sergiu Ivanov, Laurent Trilling, and Nicolas Glade. A methodology for evaluating the extensibility of Boolean networks' structure and function. In Rosa M. Benito, Chantal Cherifi, Hocine Cherifi, Esteban Moro, Luis Mateus Rocha, and Marta Sales-Pardo, editors, *Complex Networks & Their Applications IX - Volume 2, Proceedings of the Ninth International Conference on Complex Networks and Their Applications, COMPLEX NETWORKS 2020, 1-3 December 2020, Madrid, Spain*, volume 944 of *Studies in Computational Intelligence*, pages 372–385. Springer, 2020.
27. Rémi Segretain, Laurent Trilling, Nicolas Glade, and Sergiu Ivanov. Who plays complex music? On the correlations between structural and behavioral complexity measures in sign Boolean networks. In *21st IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2021, Kragujevac, Serbia, October 25-27, 2021*, pages 1–6. IEEE, 2021.

Queens of the Hill

Artiom Alhazov[1], Sergiu Ivanov[2], David Orellana-Martín[3,4]

[1]Vladimir Andrunachievici Institute of Mathematics and Computer Science,
The State University of Moldova, Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2]IBISC Laboratory, Université Paris-Saclay, Univ Évry
91020, Évry-Courcouronnes, France
E-mail : sergiu.ivanov@univ-evry.fr

[3]Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: dorellana@us.es

[4]SCORE Laboratory, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

Summary. Inspired by the programming game Core Wars, we propose in this work a framework and the organisation of king of the hill-style tournaments between P systems. We call these tournaments Queens of the Hill and the individual contestants valkyries. The goal of each valkyrie is to dissolve as many membranes of as many other valkyries as possible, while at the same time resisting the attacks. Valkyries are transition P systems with cooperative rules, target indication, and rudimentary matter–anti-matter annihilation rules. These ingredients are sufficient for computational completeness, but the context of Queens of the Hill reduces the relevance of this statement. We give some tentative examples of strategies and discuss their advantages and drawbacks. Finally, we describe how Queens of the Hill can be used as a teaching exercise, and also a tool to federate the students' creativity to push the frontiers of membrane computing.

Keywords: Core Wars, membrane dissolution, anti-matter, interaction.

1 Core Wars

To cite [11], *"Core War (or Core Wars) is a programming game where assembly programs try to destroy each other in the memory of a simulated computer."* In Core Wars, programmers design programs—called warriors—with two goals in mind:

1. kill as many other programs as possible,

2. survive for as long as possible against the attacks of the other programs.

In the most basic setup, all programs are loaded in the same shared memory space, and only feature instruction segments, i.e. their memory only contains code, and data is stored as part of some of the instructions. No memory protection is available for the instructions, so all programs can write anywhere, including to the instruction segments of competitors, which is the primary way of attacking. The simplest warrior is called the Imp and only consists of a single instruction in the special assembly language called Redcode:

```
MOV 0, 1
```

The numbers correspond to addresses in the memory space relative to the current instruction, so `0` refers to the current instruction slot, and `1` refers to the next one. This program copies its only instruction to the next memory slot, which then copies itself to the next one, etc. The Imp therefore ends up populating the whole memory with copies of itself.

As small and impressive as it is, the Imp will never actually win, because it just reproduces itself, possibly over the code of the competitors, but it never kills any competitor. To kill a process, Redcode features the special instruction `DAT`. When it is executed, the current process is killed. A simple winning code would throw `DAT` over the whole memory, while simultaneously avoiding to run this instruction in its own execution. This is what the warrior called the Dwarf does, whose detailed presentation is given in [11].

Multiple servers exist continuously running Code Wars tournaments in the king of the hill mode (see section “Climbing the hill” in [11]): 10 to 30 warriors are loaded in the same shared memory space and are run sequentially, on a single virtual processor, which interleaves the execution of the instructions of every warrior. The score of a warrior in a match roughly corresponds to the number of other warriors it has killed. The warrior with the highest score is the current king of the hill, and the warrior with the lowest score falls off the hill: it is replaced by a new warrior.

2 Queens of the Hill

In this submission we propose a framework for running king of the hill style tournaments between P systems. We refer to such tournaments as (P) *Queens of the Hill*, and we call individual contestants *valkyries*. In this section, we propose the formal framework to be used for the valkyries as well as the rules for Queens of the Hill tournaments.

2.1 Valkyries

Our choice of the P system variant for the valkyries is guided by the following two principles: ability to interact with the other contestants and ease of programming.

We choose here a variation on what is sometimes called transition P systems, which is partially inspired by P automata with matter–anti-matter annihilation rules as shown in [5] and by P colonies [2]. As a reminder, the original P automata rely on antiport rules exclusively [3].

We define a (valkyrie) P system as the following tuple:

$$\Pi = (O, \mu, w_1, \dots, w_n, R_1, \dots, R_n), \text{ where}$$

- $O = \Sigma \cup \Delta_k$ is a finite alphabet of objects,
- $\Delta_k = \{\delta_t, \bar{\delta}_t \mid 1 \leq t \leq k\} \cup \{\delta\}$ for some fixed $k \in \mathbb{N}$,
- μ is the hierarchical membrane structure bijectively labeled by the numbers from 1 to n and usually presented as a sequence of correctly nested brackets,
- w_i is the initial multiset in membrane i, $1 \leq i \leq n$,
- R_i is the finite set of rules in membrane i, $1 \leq i \leq n$.

The rules in R_i feature full cooperation and may use target indications. More precisely, a rule in R_i has the form $u \to v$, where $u \in \Sigma^\circ$, $u \neq \lambda$, is a non-empty multiset over Σ, and $v \in (O \times \mathit{Tar})^\circ$ is a multiset of symbols over O, each equipped with target indications $\mathit{Tar} = \{\mathit{in}, \mathit{here}, \mathit{out}\}$. A symbol appearing with the indication *in* in v will be sent into a non-deterministically chosen inner membrane of membrane i, a symbol with the indication *here* will remain in membrane i, and a symbol with the indication *out* will be sent to the parent membrane. If membrane i does not have any inner membranes, the symbols with target indication *in* will be kept in membrane i, i.e. the target indications *in* and *here* are equivalent in the case of elementary membranes. For readability, we will always omit the indication *here*, i.e. instead of writing $(a, \mathit{here})(a, \mathit{here})(b, \mathit{out})$ we will write $aa(b, \mathit{out})$.

The symbol $\delta \in \Delta_k$ has the special semantics of dissolving the membrane in which it appears. More formally, once δ is introduced into membrane i, all of its objects and inner membranes are moved to its parent membrane, and membrane i is removed from the system—non-elementary membrane dissolution is allowed. Membrane dissolution happens at the end of a computation step, and all introduced copies of δ are removed from the system after all dissolutions are performed. It follows incidentally that introducing any number of copies of δ in a membrane produces exactly the same as effect as introducing one copy of δ. Dissolution of the outermost (skin) membrane is forbidden, i.e. introducing a copy of δ into the skin membrane will have no effect and the symbol δ will be immediately removed.

All sets R_i, $1 \leq i \leq n$, also include the following rules:

$$\begin{aligned} R_k^\delta = {} & \{\delta_t \to \delta_{t-1} \mid 2 \leq t \leq k\} \cup \{\delta_1 \to \delta\} \\ & \cup \{\delta_t \bar{\delta}_t \to \lambda \mid 1 \leq t \leq k\}. \end{aligned}$$

Informally, the symbol δ_t is equipped with a timer which triggers the dissolution of the containing membrane after t steps. The rules $\delta_t \bar{\delta}_t \to \lambda$ have weak priority, meaning that if both a copy of δ_t and $\bar{\delta}_t$ are present, then they must be erased (annihilate), preempting the evolution rule for δ_t.

Since the left-hand sides of the rules in $R_i \setminus R_k^\delta$ are multisets over Σ, these rules cannot directly detect or rewrite the symbols in Δ_k. However, they can produce the anti-symbol $\bar{\delta}_t$ to force the annihilation of a symbol δ_t if it is present in the current membrane.

The rules are applied in the maximally parallel way, with weak priority of the annihilation rules $\delta_t\bar{\delta}_t \to \lambda$. A computation steps proceeds in the classical fashion, by first non-deterministically choosing a non-extendable multiset of rules to apply, applying it, and performing all the necessary dissolutions. A halting configuration is a configuration in which no more rules are applicable. We can consider halting computations of P systems, but due to the continual nature of the tournament, we will generally consider infinite or time-limited computations instead.

Example 1. Consider the following valkyrie P system:

$$\begin{aligned}
\Pi &= (O, [_1[_2[_3]_3]_2]_1, d, b\bar{\delta}_1, a, R_1, R_2, R_3),\\
O &= \{a, b, c, d\} \cup \Delta_2,\\
R_1 &= \{d \to d, d \to d(\delta_2, in)\} \cup R_2^\delta,\\
R_2 &= \{bc \to b\} \cup R_2^\delta,\\
R_3 &= \{a \to aa(c, out), a \to \delta\} \cup R_2^\delta.
\end{aligned}$$

As a reminder, $\Delta_2 = \{\delta_2, \delta_1, \delta\} \cup \{\bar{\delta}_1, \bar{\delta}_2\}$ and $R_2^\delta = \{\delta_2 \to \delta_1, \delta_1 \to \delta\} \cup \{\delta_2\bar{\delta}_2 \to \lambda, \delta_1\bar{\delta}_1 \to \lambda\}$. Figure 1 gives a graphical illustration of the P system above. For conciseness, we omit the rules in R_2^δ from such graphical illustrations.

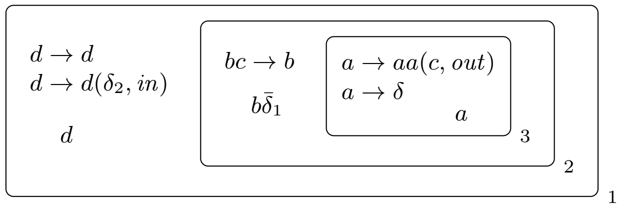

Fig. 1. A simple valkyrie P system. The rules from R_2^δ are not represented.

The rule $a \to aa(c, out)$ in membrane 3 doubles some of the a, and also ejects the corresponding number of c in membrane 2. The remaining copies of a are used to produce δ, which will dissolve membrane 3, copying all instances of a it contains into the parent membrane 2. The symbol b in membrane 2 will progressively erase all the copies of c ejected by membrane 1.

The symbol d in the skin may choose between simply maintaining itself, or also injecting δ_2 into membrane 2. The first copy of δ_2 injected into 2 will undergo the evolution rule $\delta_2 \to \delta_1$, and will afterwards annihilate with $\bar{\delta}_1$ already present there from the start. However, the second copy of δ_2 the rule $d \to d(\delta_2, in)$ will inject into membrane 2 will be free to produce δ in two steps thereby dissolving membrane 2. If by this time the rule $a \to \delta$ has not yet been applied in membrane 3, membrane 3 will become the direct inner membrane of membrane 1, so the next application of the rule $d \to d(\delta_2, in)$ will send δ_2 in membrane 3, leading to its dissolution in two steps. Therefore, this valkyrie P system always converges to a cycle of configurations in which there is only the skin membrane containing a copy of d, a copy of b, possibly some copies of c, some copies of a, as well as a symbol from $\{\delta_2, \delta_1\}$, which always ticks down to δ without any effect, since the dissolution of the skin membrane is disallowed. $\square$

2.2 Tournament Setup

The setup of Queens of the Hill tournaments is partially inspired by P colonies [2]: a set of valkyrie P systems is grouped together in a big skin membrane, which always sends back in whatever is sent out. More formally, we define an m-Queens of the Hill tournament as the following tuple:

$$\mathcal{Q} = (O, \Pi_1, \dots, \Pi_m),$$

where $O = \Sigma \cup \Delta_k$ and Π_j is a valkyrie P system as defined in Section 2.1. All P systems Π_j share the same sets of symbols Σ and O. The tournament $\mathcal{Q}$ is a P system obtained by placing all Π_j into a common outer membrane 0 with the empty initial multiset and with the following set of rules:

$$R_0 = \{a \to (a, in) \mid a \in O\} \cup R_k^\delta.$$

In other words, R_0 always sends in whatever symbols are sent out from the individual valkyries, but due to non-determinism these symbols do not necessarily end up in the valkyrie which produced them. Note that R_0 contains 2 rules for symbols δ_t: such a symbol may be sent in, or it may evolve into δ_{t-1}. As before, if the corresponding anti-symbol $\bar{\delta}_t$ is also present, the annihilation rule $\delta_t\bar{\delta}_t \to \lambda$ will have to be applied. Finally note that R_0 is the only set of rules in which the left-hand sides are allowed to include δ_t.

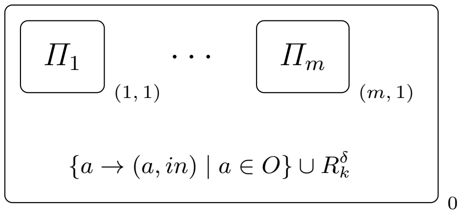

Fig. 2. An informal picture of an m-Queens of the Hill tournament $\mathcal{Q}$. The skin membranes of the valkyries Π_j are relabelled as $(j, 1)$.

A Queens of the Hill tournament obeys the same semantics as valkyrie P systems defined in Section 2.1. In particular, this means that the dissolution of the skin membranes of a valkyrie Π_j *is allowed*, because at this time it is surrounded by the bigger skin membrane of the whole tournament $\mathcal{Q}$. To preserve consistent membrane labelling, a membrane i in the valkyrie P system Π_j is renamed into membrane (j, i) in the tournament $\mathcal{Q}$.

2.3 Tournament Organization

An m-Queens of the Hill tournament runs all the valkyries in the maximally parallel mode multiple times and for a limited number of steps. At the end, the score of each valkyrie is computed from the number of its membranes that was dissolved. Non-determinism in the computations is resolved probabilistically, as it is done in the P-Lingua framework [4, 6]: at every non-deterministic branching point, one of the branches is chosen under the uniform probability distribution.

More concretely, the tournament runs in the following way:

1. Run the computation for N steps, resolving non-determinism according to the uniform probability distribution.
2. Repeat Step 1 M times.

The score of a valkyrie is computed according to the following formula:

$$\mathrm{score}(\Pi_j) = \frac{1}{|\Pi_j|}\left(|\Pi_j| - \frac{1}{M}\sum_{i=1}^{M}\mathrm{diss}_i(\Pi_j)\right),$$

where $\mathrm{diss}_i(\Pi_j)$ is the number of membranes of Π_j that were dissolved during the i-th computation (i-th run of Step 1 above), and $|\Pi_j|$ is the total number of membranes in Π_j.

Example 2. Suppose that Π_j has 5 membranes, $|\Pi_j| = 5$, and take $M = 3$. Further suppose that 2, 3, and 4 membranes of Π_j were dissolved respectively in the first, second, and third computations, i.e. $\mathrm{diss}_1(\Pi_j) = 2$, $\mathrm{diss}_2(\Pi_j) = 3$, and $\mathrm{diss}_3(\Pi_j) = 4$. Then the score of Π_j in this tournament will be:

$$\frac{1}{5}\left(5 - \frac{2+3+4}{3}\right) = \frac{2}{5}.$$

Informally, the score of a valkyrie is how many membranes on average it retains by the end of a computation of the tournament, normalized by its total number of membranes. $\square$

A valkyrie has the highest score of 1 if none of its membranes is ever dissolved in the tournament. It has the lowest score of 0 if all its membranes are always dissolved.

2.4 Tournament Parameters

Table 1 summarizes the parameters governing a Queens of the Hill tournament that were introduced in the previous sections. The values of these parameters may have a significant impact on the strategies adopted by the individual valkyries. Smaller values of $|\Sigma|$ reduce the richness of the behaviors of a valkyrie and make it less robust to perturbations coming from the skin membrane 0, i.e. from the other valkyries. Larger values of k mean more opportunities for the symbols δ_t to be captured. Larger values of m mean lower probability of receiving a symbol δ_t after emitting it into the skin membrane 0. Shorter computation lengths N mean that lightning attacks may be more feasible, while smaller values for M mean fewer computations in a tournament, which increases the contribution of randomness to the outcome.

$\lvert\Sigma\rvert$	10	The number of working symbols.
k	5	The maximal value of the index t in δ_t.
m	10–20	The number of entrants in the tournament.
N	1000	The length of a computation in the tournament.
M	50	The total number of computations in the tournament.

Table 1. A summary of the parameters governing a Queens of the Hill tournament, together with the possible values for these parameters.

3 A Note on Computational Complexity

Valkyrie P systems as defined in Section 2 are quite obviously computationally complete, even with a subset of the ingredients. In particular, full cooperation together with the maximally parallel mode suffice to simulate arbitrary register machines. We refer the reader to the first publication in membrane computing [15] for the very first proofs, as well as to the more recent [10, 16] for a sample of the wide variety of techniques for proving computational completeness of P system variants. For the record and for the sake of the discussion of the possible strategies in Queens of the Hill tournament, we briefly recall a proof of computational completeness of P systems as defined above.

A (deterministic) register machine is an abstract computational device consisting essentially of a finite set of registers and a program. The registers can contain natural numbers or zero. The program consists of the following two types of instructions:

- $(p, \mathrm{ADD}(r), q)$: in state p, increment register r and go to state q;
- $(p, \mathrm{SUB}(r), q, s)$: in state p, check the value of register r; if its value is strictly positive, decrement it and go to state q; otherwise go to state s.

Register machines are famously computationally complete. We refer the reader to [13] for a much more in-depth discussion.

One-membrane valkyrie P systems can simulate both types of register machine instructions, even without dissolution or anti-matter rules. Classically, the alphabet Σ will include one symbol per state p of the register machine, and the value of register r will be represented by the multiplicity of symbol a_r. The instruction $(p, \mathrm{ADD}(r), q)$ can be directly simulated by the rule $p \to q a_r$. The simulation of $(p, \mathrm{SUB}(r), q, s)$ is more intricate, as usual, and relies on non-determinism and maximal parallelism: the symbol p non-deterministically guesses whether the register is empty, and a trap symbol is produced if the guess is wrong. The following table lists the rules for both branches, arranged by steps:

	Decrement	*Zero test*
1.	$p \to \bar{p}_1 \hat{p}_1$	$p \to \tilde{p}_1 \dot{p}_1$
2.	$\bar{p}_1 a_r \to \bar{p}_2,\ \hat{p}_1 \to \hat{p}_2$	$\dot{p}_1 a_r \to \#,\ \tilde{p}_1 \to \tilde{p}_2$
3.	$\hat{p}_2 \bar{p}_2 \to q,\ \hat{p}_2 \bar{p}_1 \to \#$	$\tilde{p}_2 \dot{p}_1 \to s$

The decrement branch begins by splitting the state symbol p into $\bar{p}_1$ and $\hat{p}_1$. The symbol $\bar{p}_1$ erases a copy of a_r if it is present in the system and evolves into $\bar{p}_2$. It does not evolve if no copies of a_r are present. At the same time, $\hat{p}_1$ evolves into $\hat{p}_2$. In the third step, $\hat{p}_2$ evolves into q in the presence of $\bar{p}_2$, i.e. in the case in which the decrement was successful. If the decrement could not happen, $\hat{p}_2$ finds $\bar{p}_1$, which produces the trap symbol.

The zero test branch begins by splitting the state symbol p into $\tilde{p}_1$ and $\dot{p}_1$. The symbol $\dot{p}_1$ must evolve into the trap symbol $\#$ if it finds a copy of a_r, as $\dot{p}_1 a_r \to \#$ is the only rule which may transform $\dot{p}_1$. In the meantime, $\tilde{p}_1$ evolves into $\tilde{p}_2$. If in the third step $\dot{p}_1$ is still present in the system, this means that it did not find any copies of a_r, the register is empty, and the symbol s is produced. Otherwise $\tilde{p}_2$ cannot evolve, but this also means that a trap symbol was produced at step 2, meaning that the computation will never halt.

The argument above shows that the language of valkyries in Queens of the Hill tournaments is rich enough. However, note how this argument relies on two essential details which are partially relevant or even irrelevant in Queens of the Hill: non-determinism and halting. On the one hand, non-determinism is resolved probabilistically, meaning that not all possibilities will be explored, and that some of them may be explored multiple times. Furthermore, proofs of computational completeness in P systems classically consider the results produced at the end of halting computations, while in Queens of the Hill halting does not have a central role. What is important in Queens of the Hill is communicating with the other valkyries, i.e. attempting to dissolve as many of their membranes as possible, as soon as possible. From this standpoint, efficiency is important, while actual computational complexity is much less relevant, as long as the valkyrie manages to attain a relatively high score. Finally, note how $|\Sigma|$ is a powerful tool for modulating the complexity and the efficiency of individual valkyries.

4 Tentative Strategies

The main goal of Queens of the Hill is turning P system design into a game involving teams of students on the front line, backed by researchers collecting and systematizing the explicit and implicit knowledge produced by the teams designing the valkyries. In this section, we present several tentative strategies, whose efficiency or relevance will be the subject of immediate future work.

One of the first strategies one may think of when seeing the rules of Queens of the Hill is the Bomber: eject δ_t for some value of t into the skin membrane 0 and hope that none of those symbols is sent back into the same membrane. The

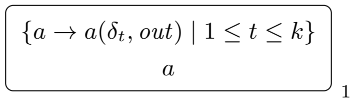

Fig. 3. The Bomber.

efficiency of the bomber decreases as the number of valkyries decreases. For example, when there is only one other valkyrie, the probability is quite high that the ejected δ_t lands back in the Bomber. Note that this probability is not exactly $\frac{1}{2}$, since the rule $\delta_t \to \delta_{t-1}$ can also be applied in the skin, potentially until the production of δ.

The Bomber can be made more robust by making it accumulate copies of $\bar{\delta}_t$ for some values of t, so that the symbols δ_t coming from the skin annihilate with the corresponding copies of $\bar{\delta}_t$. While this strategy can deal with an occasional $\bar{\delta}_t$, it will be quickly overwhelmed when sharing the tournament with a considerable population of bombers.

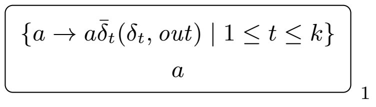
$\{a \to a\bar{\delta}_t(\delta_t, out) \mid 1 \leq t \leq k\}$
a
1

Fig. 4. The Bar Bomber.

Another variation of the Bomber is the strategy of ejecting $\bar{\delta}_t$ in the hope of neutralizing δ_t before it even gets into the valkyrie. This has the obvious disadvantage that it will also protect the other valkyries from δ_t.

In case the number of competing valkyries m—or an upper bound on m—is known, robustly dealing with such bomber strategies is in fact not very difficult: it suffices to ensure the presence of $r(m-1)$ copies of $\bar{\delta}_1$ at all times, where $r \in \mathbb{N}\setminus\{0\}$ is a natural factor which we discuss in the following paragraph. Indeed, it is not necessary to provide for $\bar{\delta}_t$ for $t > 1$, as these symbols will inextricably tick down to δ_1 and will have to annihilate with $\bar{\delta}_1$.

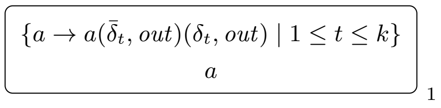
$\{a \to a(\bar{\delta}_t, out)(\delta_t, out) \mid 1 \leq t \leq k\}$
a
1

Fig. 5. The Anti-Bomber.

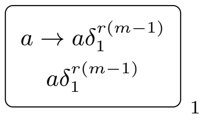

Fig. 6. The Delta Wall.

The idea behind the factor r is that other strategies may try to beat the Delta Wall by having rules emitting a large number of δ_t. However, the more such symbols are emitted, the lower the probability that they end up in the same valkyrie, meaning that the Delta Wall will have a high degree of resilience, even for smaller values of r, like 3 or even 2.

Another protective strategy consists in wrapping the valkyrie in a couple of additional membranes. In this way, the valkyrie can tolerate several membrane dissolutions without being thrown out of the game.

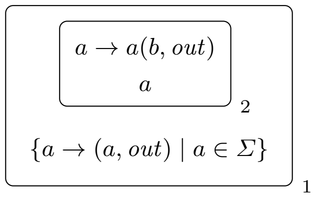

Fig. 7. The 2-layer Onion.

Remark that the Onion will have trouble emitting δ_t symbols. Firstly, the rules in the valkyrie are not allowed to contain δ_t in their left-hand sides, so the rules of the shape $\delta_t \to (\delta_t, out)$ are not allowed. If instead of emitting δ_t a different symbol d is used, then it is necessary to convert d into δ_t at some moment. If such conversion rules only appear in the outermost membrane of the Onion, then dissolving that membrane will remove those rules. On the other hand, including such rules in every layer of the Onion will create the possibility that an inner level inadvertently causes the dissolution of an outer level. Therefore, a strategy which may work best with the Onion would consist in relying on the relative scarcity of the symbols in Σ and in trying to destabilize the other valkyries by forcing some unexpected symbols into their membranes. For this to work, $|\Sigma|$ has to be sufficiently small.

Finally, the last tentative strategy we present in this section is the Bombshell. The idea is to have multiple inner membranes which are all released into the skin membrane of the tournament, therefore creating a family of cooperating agents belonging to the same team. This allows for exceeding the total number of valkyies m, but comes at the price of dissolving a membrane, which will be reflected in the final score.

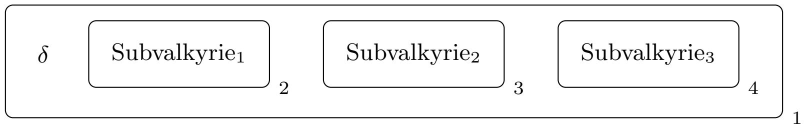

Fig. 8. The scheme of a 3-charge Bombshell.

5 Future Work and Perspectives

The immediate future work is setting up Queens of the Hill tournaments between valkyries designed by teams of students taking a course in formal languages or in natural computing. Queens of the Hill can be seen as a programming exercise in the language of an unconventional model of computing with a concrete goal: attacking all other contestants and surviving against their attacks for as long as possible. This context can also be used to introduce questions from theoretical biology about evolution and robustness, somewhat in the spirit of [17, 18]. We remark that such exercises are quite widespread in teaching of multi-agent systems and autonomic systems, as NetLogo-related resources illustrate [19].

To us as teachers and researches (enseignants-chercheurs as they say in French), Queens of the Hill is a great opportunity to employ our students' creativity to push the frontiers of what can be done with P systems. In the particular setup we describe in this paper we focus on transition P systems with non-elementary membrane dissolution and some $r$udimentary matter-antimatter annihilation rules, a model directly supported by P-Lingua. Obviously, other variants of P systems and the corresponding simulator engines can be used as the underlying formalism, thereby stimulating the students' interest in these other variants. Among the salient examples we cite kernel P systems [7, 12] and cP systems [8, 9, 14].

While valkyrie P systems are in principle computationally complete (Section 3), individual computational steps are less expressive than register machine instructions, meaning that designing valkyries *de facto* explores the capabilities of a less powerful language. Furthermore, good valkyrie design will require estimating the probabilities of different branches of computation, which will encourage the students to delve deeper into probability theory.

The setup we propose in this paper is at an early stage. We will most likely need to further tune the values of the parameters in Table 1, and probably also adjust some aspects of the definitions of valkyrie P systems as well as of the tournament in order to avoid trivial edge cases and incite the design of complex strategies. An important question is the relevance of the scoring function score(Π_j) introduced in Section 2.3—other scoring functions may better capture the results of the competition. It is also possible to define scoring functions measuring the production of a certain set of symbols, thereby shifting the focus away from membrane dissolution entirely. One could also think about tracking the origins of the symbols, which could in principle allow saying which valkyrie dissolved which other valkyrie. This would require a rather fine analysis of the computations.

On a final note, we remark that while Queens of the Hill tournaments are directly inspired by Core Wars, the P system context shuffles things up quite a bit. In particular, data is secondary in Core Wars, and warriors interact by writing over each other's code. If we take the rules to be the program in P systems, then the programs of the valkyries are immutable in the sense that individual rules cannot be modified[1]. However, it is possible to instantly and entirely erase parts of their programs by dissolving the corresponding membranes. Furthermore, P systems are inherently non-deterministic, which we translate into a probabilistic framework, while warriors in Core Wars are deterministic. These remarks make us believe that Queens of the Hill tournaments have great potential waiting to be explored.

Acknowledgements

The authors would like to thank the Organizing Committee of the 19th Brainstorming Week on Membrane Computing[2] (BWMC 2023) for organizing this fruitful event.

References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Polymorphic P systems: A survey. Technical report, Bulletin of the International Membrane Computing Society, December 2016.
2. Lucie Ciencialová, Erzsébet Csuhaj-Varjú, Ludek Cienciala, and Petr Sosík. P colonies. *J. Membr. Comput.*, 1(3):178–197, 2019.
3. Erzsébet Csuhaj-Varjú and György Vaszil. P automata or purely communicating accepting P systems. In Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2002.
4. Ignacio Pérez-Hurtado et al. The P-Lingua Website. `http://www.p-lingua.org/wiki/index.php/Main_Page`, Retrieved in May 2023.

[1] This may be an opportunity for plugging in polymorphic P systems and other P system variants with dynamic rules [1].

[2] `http://www.gcn.us.es/19bwmc`

5. Rudolf Freund, Sergiu Ivanov, and Ludwig Staiger. Going beyond turing with P automata: Regular observer ω-languages and partial adult halting. *Int. J. Unconv. Comput.*, 12(1):51–69, 2016.
6. Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Miguel A. Martínez-del-Amor, Enrique Orejuela-Pinedo, and Ignacio Pérez-Hurtado. P-lingua 2.0: A software framework for cell-like P systems. *Int. J. Comput. Commun. Control*, 4(3):234–243, 2009.
7. Marian Gheorghe, Rodica Ceterchi, Florentin Ipate, Savas Konur, and Raluca Lefticaru. Kernel P systems: From modelling to verification and testing. *Theor. Comput. Sci.*, 724:45–60, 2018.
8. Alec Henderson and Radu Nicolescu. Actor-like cP systems. In Thomas Hinze, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing - 19th International Conference, CMC 2018, Dresden, Germany, September 4-7, 2018, Revised Selected Papers*, volume 11399 of *Lecture Notes in Computer Science*, pages 160–187. Springer, 2018.
9. Alec Henderson, Radu Nicolescu, and Michael J. Dinneen. Solving a PSPACE-complete problem with cP systems. *J. Membr. Comput.*, 2(4):311–322, 2020.
10. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
11. Ilmari Karonen. The beginners' guide to Redcode. `https://vyznev.net/corewar/guide.html`, version 1.23, August 11, 2020.
12. Savas Konur, Laurentiu Mierla, Florentin Ipate, and Marian Gheorghe. kPWorkbench: A software suit for membrane systems. *SoftwareX*, 11:100407, 2020.
13. Ivan Korec. Small universal register machines. *Theor. Comput. Sci.*, 168(2):267–301, 1996.
14. Radu Nicolescu and Alec Henderson. An introduction to cP systems. In Carmen Graciani Díaz, Agustín Riscos-Núñez, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*, volume 11270 of *Lecture Notes in Computer Science*, pages 204–227. Springer, 2018.
15. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
16. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
17. Rémi Segretain, Sergiu Ivanov, Laurent Trilling, and Nicolas Glade. A methodology for evaluating the extensibility of boolean networks' structure and function. In Rosa M. Benito, Chantal Cherifi, Hocine Cherifi, Esteban Moro, Luis Mateus Rocha, and Marta Sales-Pardo, editors, *Complex Networks & Their Applications IX - Volume 2, Proceedings of the Ninth International Conference on Complex Networks and Their Applications, COMPLEX NETWORKS 2020, 1-3 December 2020, Madrid, Spain*, volume 944 of *Studies in Computational Intelligence*, pages 372–385. Springer, 2020.
18. Rémi Segretain, Laurent Trilling, Nicolas Glade, and Sergiu Ivanov. Who plays complex music? On the correlations between structural and behavioral complexity measures in sign Boolean networks. In *21st IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2021, Kragujevac, Serbia, October 25-27, 2021*, pages 1–6. IEEE, 2021.
19. Uri Wilensky. NetLogo. `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

A solution to the only one object problem with dissolution rules

Julien Caselmann[1], David Orellana-Martín[2,3]

[1]Humboldt-Universität zu Berlin
E-mail: caselmaj@informatik.hu-berlin.de
[2]Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: dorellana@us.es
[3]SCORE Laboratory, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

Summary. In membrane computing, it is usual to obtain solutions to decision problems by means of (non-)uniform families of membrane systems, where each P system of the family can solve one or more than one instance of the problem. In this work, a new solution to the `ONLY-ONE-OBJECT` problem is provided by means of a single membrane system, that is capable of solving each instance of the problem.

Key words: Membrane Computing, computational complexity theory, only one object problem.

1 Introduction

Membrane Computing is a relatively young compared to other areas of computer science research and therefore still presents us with some open questions. Some computational models inspired by biological cells have already been very successful. The ideas of this computational framework are materialized into non-deterministic, parallel and distributed models of computation called membrane systems (or P systems). Certain membrane systems have achieved surprising results, ranging from a simple computation of a square number [1] to an *ad-hoc* solution for the `SAT` problem [2]. It is still not entirely clear what exactly these membrane systems are capable of and what is beyond their reach. In this sense, several works concerning lower and upper bounds of different classes of membrane systems have been published [3, 4]. . Numerous questions arise from that, including the famous 26 problems of Păun [5]. $\mathbf{PMC}_{\mathcal{R}}$ is the class of problems solvable in polynomial time by membrane systems of the class $\mathcal{R}$. In this area, a new way to tackle the **P** *versus* **NP** problem appears in the form of efficiency frontiers. As **P** *versus* **NP** is one of the most exciting problems in complexity theory in computer science, it is of utmost interest to find out what the connections between the classes **PMC** and **P**, **NP** are.

In [6, 7, 8, 9], a new methodology for solving decision problems or demonstrating the non-solvability of a problem by means of a single membrane system is presented. On the one hand, a solution to the `PARITY` problem is given by means of a single membrane system using transition P systems using dissolution rules and only two objects in the left-hand side of evolution rules. On the other hand, it is demonstrated that the `ONLY-ONE-OBJECT` problem cannot be solved by means of a single P system with active membranes without polarizations and without dissolution rules.

The rest of the work is structured as follows. In the following sections, we present some preliminaries about languages and set theory and we recall the model of recognizer polarizationless P systems with active membranes with dissolution rules. In Section 4, we briefly recall the concept of complexity classes with single recognizer membrane systems. Section 5 is devoted to cite some of the main uses of the dependency graph technique. Next, we propose a solution to the `ONLY-ONE-OBJECT` by means of a single recognizer membrane system from $\mathcal{NAM}^0(+d, -ne)$. We finish the paper with some conclusions and interesting open research lines.

2 Preliminaries

In this section, we introduce some basic concepts and recognizer polarizationless P systems with active membranes with dissolution rules. For deeper questions concerning formal languages and membrane systems, we refer the reader to [10, 11].

An *alphabet* Γ is a non-empty set and their elements are called *symbols*. A *string* u over Γ is an ordered finite sequence of symbols, that is, a mapping from a natural number $n \in \mathbb{N}$ onto Γ. The number n is called the *length* of the string u and it is denoted by $|u|$. The empty string (with length 0) is denoted by λ. The set of all strings over an alphabet Γ is denoted by Γ^*. A *language* over Γ is a subset of Γ^*.

A *multiset* over an alphabet Γ is an ordered pair (Γ, f) where f is a mapping from Γ onto the set of natural numbers $\mathbb{N}$. The *support* of a multiset $m = (\Gamma, f)$ is defined as $supp(m) = \{x \in \Gamma \mid f(x) > 0\}$. A multiset is finite (respectively, empty) if its support is a finite (respectively, empty) set. We denote by $\emptyset$ the empty multiset. Let $m_1 = (\Gamma, f_1)$, $m_2 = (\Gamma, f_2)$ be multisets over Γ, then the union of m_1 and m_2, denoted by $m_1 + m_2$, is the multiset (Γ, g), where $g(x) = f_1(x) + f_2(x)$ for each $x \in \Gamma$. We denote by $M(\Gamma)$ the set of all multisets over Γ.

3 Polarizationless P systems with active membranes

First presented in [1], a P system is a computational model inspired by a biological cell.

Definition 1. *A polarizationless P system with active membranes without division rules of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$$

where:

- *Γ is a finite (working) alphabet whose elements are called objects.*
- *H is a finite alphabet such that $H \cap \Gamma = \emptyset$ whose elements are called labels.*
- *$q \geq 1$ is the degree of the system.*
- *μ is a labelled rooted tree (called membrane structure) consisting of q nodes injectively labelled by elements of H (the root of μ is labelled by r_μ).*
- *$\mathcal{M}_1, \dots, \mathcal{M}_q$ are multisets over $\Gamma \setminus \Sigma$.*
- *$\mathcal{R}$ is a finite set of rules, of the following forms:*
  (*a*) $[\, a \to u \,]_h$, *where* $h \in H$, $a \in \Gamma$, $u \in M(\Gamma)$, (object evolution rules).

  (*b*) $a\,[\quad]_h \to [\, c \,]_h$, *where* $h \in H \setminus \{r_\mu\}$, $a, b, c \in \Gamma$ (send-in communication rules).

  (*c*) $[\, a \,]_h \to b\,[\quad]_h$, *where* $h \in H$, $a, b \in \Gamma$ (send-out communication rules).

  (*d*) $[\, a \,]_h \to b$, *where* $h \in H \setminus \{i_{out}, skin\}$, $a, b \in \Gamma$ (dissolution rules).
- *$i_{out} \in H \cup \{env\}$.*

A polarizationless P system with active membranes

$$\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{out})$$

can be viewed as a set of q membranes, labelled by elements of H, arranged in a hierarchical structure μ given by a rooted tree (called membrane structure) whose root is called the *skin membrane*, such that: (a) $\mathcal{M}_1, \dots, \mathcal{M}_q$ represent the finite multisets of *objects* initially placed in the q membranes of the system; (b) $\mathcal{R}$ is a finite set of rules over Γ associated with the labels; and (c) $i_{out} \in H \cup \{env\}$ indicates the output region. We use the term *region* i to refer to membrane i in the case $i \in H$ and to refer to the "environment" of the system in the case $i = env$.

A membrane that does not have internal membranes (i.e. it is a leaf of the tree structure μ) is called an elementary membrane. Otherwise, it is considered a non-elementary membrane. The membrane that surrounds the whole system is called *skin* membrane.

An *instantaneous description* or a *configuration* $\mathcal{C}_t$ at an instant t of a polarizationless P system with active membranes is described by the following elements: (a) the membrane structure at instant t, and (b) all multisets of objects over Γ associated with all the membranes present in the system at that moment. The initial configuration of Π is described as $\mathcal{C}_0 = (\mu, \mathcal{M}_1, \ldots, \mathcal{M}_q; \emptyset)$. We denote the contents of the region h in the moment t as $\mathcal{C}_t(h)$.

An object evolution rule $[\, a \to u \,]_h$ for $h \in H$, $a \in \Gamma$, $u \in M(\Gamma)$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$ which contains object a. When applying such a rule, object a is consumed and objects from multiset u are produced in that membrane.

A send-in communication rule $a\,[\quad]_h \to [\, b \,]_h$ for $h \in H$, $a, b \in \Gamma$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$ such that h is not the label of the root of μ and its parent membrane contains object a. When applying such a rule, object a is consumed from the parent membrane and object b is produced in the corresponding membrane h.

A send-out communication rule $[\, a \,]_h \to b\,[\quad]_h$ for $h \in H$, $a, b \in \Gamma$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$ such that it contains object a. When applying such a rule, object a is consumed from such membrane h and object b is produced in the parent of such membrane.

A dissolution rule $[\, a \,]_h \to b$ for $h \in H \setminus \{i_{out}\}$, $a, b \in \Gamma$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$, different from the skin membrane and the output region, such that it contains object a. When applying such a rule, object a is consumed, membrane h is dissolved and its objects are sent to the parent (or the first ancestor that has not been dissolved).

A computational step is made by the application of the aforementioned rules in the following way:

- One object can fire only one rule;
- Object evolution rules can fire in a maximal parallel way; that is, all the evolution rules that can be fired will be fired;
- In each membrane, only one rule of the types (b), (c) or (d) can be fired in each computational step;
- A computational step is divided in two microsteps: First, all the transformations of objects are made, and second, membranes that must be dissolved will be dissolved.

If an object can fire more than one rule, then it will select one of them non-deterministically. $\mathcal{C}_t$ leads to $\mathcal{C}_{t+1}$ if the latter can be obtained from the former by applying the rules in the previously explained way, and it is denoted by $\mathcal{C}_t \Rightarrow_\Pi \mathcal{C}_{t+1}$. A computation of the system is a sequence of configurations $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_n)$, where $\mathcal{C}_0$ is the initial configuration of the membrane system, and for each $\mathcal{C}_t, t \geq 1$, $\mathcal{C}_{t-1} \Rightarrow_\Pi \mathcal{C}_t$. We say that the computation is finite if $n \in \mathbb{N}$.

In [12, 13], a special type of membrane systems is introduced in order to solve decision problems, the well-known recognizer membrane systems. A recognizer membrane system is a membrane system from any class of P systems (e.g., cell P systems, tissue P systems and so on) that has special requirements.

Definition 2. *A recognizer polarizationless P system with active membranes without division rules of degree $q \geq 1$ is a tuple*

$$(\Pi, \Sigma, i_{in})$$

where:

- $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ *is a polarizationless P system with active membranes without division rules of degree* $q \geq 1$ *where:*
  - Γ *has two special symbols,* `yes` *and* `no`.
  - $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are multisets over* $\Gamma \setminus \Sigma$.
  - $i_{out} = env$ *is the environment of the system.*
- $\Sigma \subsetneq \Gamma$.
- $i_{in} \in H$ *is the input membrane.*

Let $m \in M(\Sigma)$. We denote by $\Pi + m$ the membrane system Π with input m; that is, the membrane system Π where the multiset m is introduced in the initial configuration in the input membrane i_{in}. Then, the initial configuration of $\Pi + m$ is $\mathcal{C}_0 = (\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} + m, \ldots, \mathcal{M}_q; \emptyset)$. We recall the concept of solvability of decision problems by means of recognizer membrane systems:

Definition 3. *Let* $X = (I_X, \theta_X)$ *be a decision problem, and let* $cod : I_X \to M(\Sigma)$ *be a function that transforms an instance of* X *to a multiset over* Σ*, that will be the input of* Π. *We say that* Π *solves an instance* $u \in I_X$ *of the decision problem* X *if:*

- *The system* $\Pi + cod(u)$ *sends only one object* `yes` *or one object* `no`*, but not both, to the environment, and only in the last step of the computation; and*
- *The system* $\Pi + cod(u)$ *is* ***confluent****; that is, all the computations halt and return the same result.*

4 The complexity classes $\mathbf{PMC}_{\mathcal{R}}$ and $\mathbf{PMC}^{1f}_{\mathcal{R}}$

We recall the definition of various computational complexity classes in the framework of membrane computing.

Definition 4. *Let* $\mathcal{R}$ *be a class of recognizer membrane systems. We say that a family of recognizer membrane systems* $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ *from* $\mathcal{R}$ *solves a decision problem* $X = (I_X, \theta_X)$ *in a uniform way if the following hold:*

- *There exists a pair of functions* (cod, s) *computable in polynomial time over* I_X *such that* $cod(u) \in M(\Sigma)$ *(input multiset) and* $s(u) \in \mathbb{N}$ *(size of the instance);*
- *For each* $n \in \mathbb{N}$, $s^{-1}(n) \subseteq I_X$.
- $\mathbf{\Pi}$ *is polynomially bounded, sound and complete with regard to* (X, cod, s).

In this way, for solving a certain instance $u \in I_X$, we need to know the answer of the membrane system $\Pi(s(u)) + cod(u)$. For more detail, we refer the reader to [12].

In [9, 6], authors introduce the complexity class $\mathbf{PMC}^{1f}_{\mathcal{R}}$ as a manner to deal with decision problems with a single recognizer membrane system.

Definition 5. *Let $\mathcal{R}$ be a class of recognizer membrane systems. Let $X = (I_X, \theta_X)$ be a decision problem such that I_X is a language over a finite alphabet Σ_X. We say that problem X is solvable in polynomial time by a single membrane system Π from $\mathcal{R}$ free of external resources, denoted by $X \in \boldsymbol{PMC}^{1f}_{\mathcal{R}}$, if the following hold:*

- *The input alphabet of Π is Σ_X.*
- *The system Π is polynomially bounded, sound and complete with regard to X*

The term *free of resources* means that the input is directly introduced as a multiset from the instance, without "being encoded".

The class $\mathbf{PMC}^{1p}_{\mathcal{R}}$ is also defined in the aforementioned paper, being defined in a similar way to $\mathbf{PMC}^{1f}_{\mathcal{R}}$, but in this case the encoding of the input instance is allowed.

From these asserts, it is trivial to see that $\mathbf{PMC}^{1f}_{\mathcal{R}} \subseteq \mathbf{PMC}^{1p}_{\mathcal{R}} \subseteq \mathbf{PMC}_{\mathcal{R}}$.

5 The Origins of the Dependency Graph

The non-deterministic nature of membrane systems let a membrane system have, instead of a single computation, a tree of computations, usually denoted by $Comp(\Pi)$. The dynamics of the system are captured by $Comp(\Pi)$, being the initial configuration the root node of the tree and there exists an edge between $\mathcal{C}$ and $\mathcal{C}'$ if and only if $\mathcal{C} \Rightarrow_{\Pi} \mathcal{C}'$. The configuration paths of maximum length up to a leaf are the computations of Π and a computation terminates if and only if the path is finite. From the definition of $\mathbf{PMC}_{\mathcal{R}}$ we know that the obtained membrane systems must be confluent. Therefore, it is sufficient to consider only one computation per problem case. An interesting question would consist on finding the computational path of minimum length, but for this one has to measure the degree of similarity between two configurations (e.g., by the distance metric [14]). In this context, the dependency graph was introduced, which represents the dependencies between the membrane states (configurations) and the set of rules of the P system.

6 The Dependency Graph as a Proof Technique

From that very first application, different applications that are not related to the first one appeared.

6.1 Non-Efficiency of specific Membrane Systems

Let us now consider polarizationless recognizer P systems with active membranes that do not use dissolution rules. The dependency graph G is a directed graph whose vertex set consists of the initial configuration and all membrane configurations (a, h) (= alphabet object and membrane label on which it appears), for which

a appears on the right or left side of a rule. If there is a rule leading from configuration 1 to configuration 2, then nodes 1 and 2 in G are connected by an edge. Here, they use the concept of "accessibility" in the graph. The objects initially placed in the system can be considered as the initial nodes (taking into account the corresponding membranes). Instead of simulating the whole system, the graph can be constructed from the definition of the recognizer membrane system and the question is transformed to the following one:

> Is there a path from the initial nodes to the node (yes, env)?

In [15], the well-known Păun's conjecture was stated. Is $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(+d,-ne)}$. In [16], they use the concept to give a partial affirmative answer for the case where dissolution is forbidden and division rules both for elementary and non-elementary membranes. In that work, it is shown that $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,+ne)}$.

As stated above, we need to find a path from the initial nodes to the node (yes, env). But for this purpose, we reduce the entire problem to the `REACHABILITY` problem, that is stated as follows:

> Given a directed graph $G = (V, E)$ and two vertices $s, t \in V$. Is there a path from s to t?

It is known that `REACHABILITY` $\in \mathbf{P}$, thus completing the proof.

6.2 Negative Results in Membrane Computing

The dependency graph can be used to show negative results as well. For instance, on can show:

$$\texttt{ONLY-ONE-OBJECT} \notin \mathbf{PMC}^{1f}_{\mathcal{AM}^0(-d,+ne)} \tag{1}$$

In this context, a computation is accepting if and only if there is a path in G from s to t, with s being the initial and t the final vertex of the computation. We will show (1) by contradiction. Let us assume there were such a P system $\Pi \in \mathcal{AM}^0(-d, +ne)$. Then, a path from the initial vertex to the vertex (yes, env) that passes the vertex (a, i_{in}) would exist, to correctly resolve the case $\{a\}$. When we analyze the case $\{a^n\}, n > 1$, the following holds:

$$\forall n > 1 : G_{\Pi+\{a\}} = G_{\Pi+\{a^n\}}$$

This holds, because the vertex set of the dependency graph is a **set** and not a multiset. And because of that, a different multiplicity of the same element (n elements a instead of a single a) does not change the graph at all. Thus, every computation would be accepting $\Rightarrow$ this contradicts our assumption.

7 Proposed Solution for the `ONLY-ONE-OBJECT` Problem Using Dissolution

As we have just seen, the `ONLY-ONE-OBJECT` problem cannot be solved with a P system $\Pi \in \mathcal{AM}^0(-d, +ne)$. However, if we allow the use of dissolution rules, the problem is solvable. In Figure 1, we propose a P system that is in $\mathcal{AM}^0(+d, -ne)$ and solves `ONLY-ONE-OBJECT`. The system takes a multiset $\{a^n\}, n \in \mathbb{N}^+$ as input in membrane 3 and outputs the answer to the problem (*yes* or *no*) into the system's surrounding. The system uses three dissolution rules: one in membrane 3 and two in membrane 2, two of these only activate themselves if there is an a present in the membrane. In the case $n = 1$, only membrane 3 will be diluted by those rules, in the other case ($n > 1$), the membranes 3 and 2 are diluted. Making use of the auxiliary element β, we can now "count" the number of necessary computation steps to reach membrane 1. If there is more than one a, membranes 3 and 2 will be diluted in two computation steps, so the auxiliary element β'' will be in membrane 1 and the system will output *no*. On the other hand, if there is only one a, membrane 2 won't be diluted using the dissolution rule $[a]_2 \rightarrow \delta$, but using this one instead: $[\beta''']_2 \rightarrow \delta$. So, if there is only one a, we can increment the auxiliary element β until it reaches β'''. In the final step, if there is a β''' in membrane 1, we output *yes*. It is impossible by design that are a β'' and a β''' at the same time in membrane 1, which means that the system will always output the same and the correct answer.

8 Conclusions

Multiple use cases exist for the dependency graph in the analysis of computational models. With that tool, we were able to show the non-efficiency of a membrane system and the non-existence of a solution for a given problem when using a specific membrane system (`ONLY-ONE-OBJECT` $\notin \mathbf{PMC}^{1f}_{\mathcal{AM}^0(-d,+ne)}$). Furthermore, we presented a P system in $\mathcal{AM}^0(+d, -ne)$ that solves the `ONLY-ONE-OBJECT` problem.

If we want to solve **P** *versus* **NP**, we should keep making efforts in exploring new techniques to analyze the possibility and feasibility in the context of membrane systems.

Acknowledgements

D. Orellana-Martín acknowledges Contratación de Personal Investigador Doctor. (Convocatoria 2019) 43 Contratos Capital Humano Línea 2. Paidi 2020, supported by the European Social Fund and Junta de Andalucía.

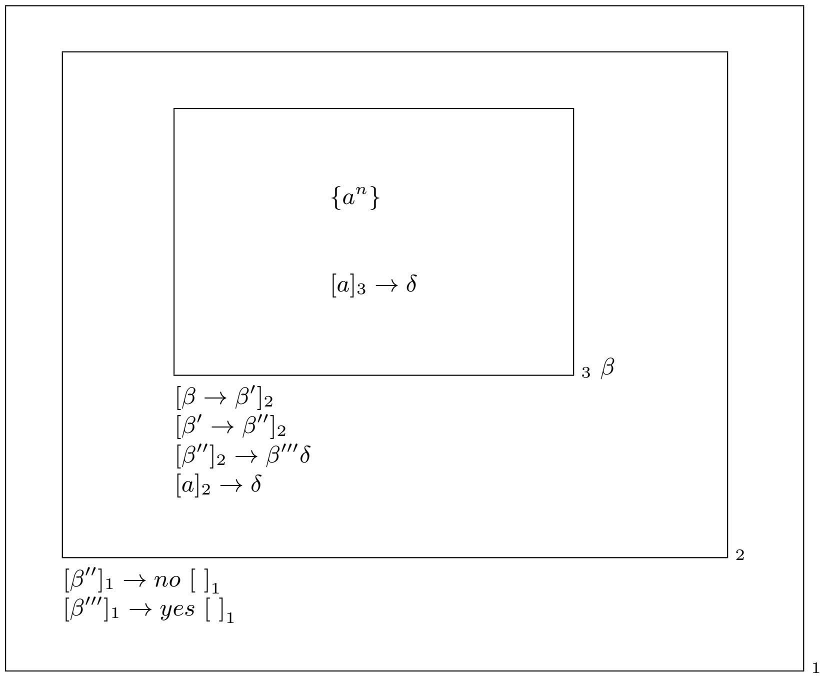

Fig. 1. P system $\Pi \in \mathcal{AM}^0(+d, -ne)$ solving `ONLY-ONE-OBJECT`

References

[1] G. Păun. "Computing with Membranes". In: *Journal of Computer and System Sciences* 61.1 (2000), pp. 108–143. ISSN: 0022-0000. DOI: `https://doi.org/10.1006/jcss.1999.1693`.

[2] G. Păun. "P Systems with Active Membranes: Attacking NP-Complete Problems". In: *J. Autom. Lang. Comb.* 6.1 (Jan. 2001), pp. 75–90. ISSN: 1430-189X.

[3] B. Song, K. Li, D. Orellana-Martín, M. J. Pérez-Jiménez, and I. Pérez-Hurtado. "A Survey of Nature-Inspired Computing: Membrane Computing". In: *ACM Comput. Surv.* 54.1 (Feb. 2021). ISSN: 0360-0300. DOI: `10.1145/3431234`.

[4] G. Mauri, A. Leporati, L. Manzoni, A. E. Porreca, and C. Zandron. "Complexity Classes for Membrane Systems: A Survey". In: *Language and Automata Theory and Applications.* Ed. by A.-H. Dediu, E. Formenti, C. Martín-Vide, and B. Truthe. Cham: Springer International Publishing, 2015, pp. 56–69. ISBN: 978-3-319-15579-1.

[5] G. Păun. "Introduction to Membrane Computing". In: *The Journal of Logic and Algebraic Programming* 79 (Jan. 2006), pp. 1–42. DOI: `10.1007/3-540-29937-8_1`.

[6] D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez, and M. J. Pérez-Jiménez. “A new perspective on computational complexity theory in Membrane Computing”. In: *BWMC2019: 17th Brainstorming Week on Membrane Computing.* 2019, pp. 117–126.

[7] D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez, and M. J. Pérez-Jiménez. “An apparently innocent problem in Membrane Computing”. In: *BWMC2019: 17th Brainstorming Week on Membrane Computing).* IMCS: International Membrane Computing Society. 2019, pp. 127–138.

[8] L. Valencia-Cabrera, D. Orellana-Martín, I. Pérez-Hurtado, and M. J. Pérez-Jiménez. “New applications for an old tool”. In: *BWMC2019: 17th Brainstorming Week on Membrane Computing.* 2019, pp. 165–170.

[9] L. Valencia-Cabrera, D. Orellana-Martín, I. Pérez-Hurtado, and M. J. Pérez-Jiménez. “Dependency Graph Technique Revisited”. In: *CMC20: 20th International Conference on Membrane Computing (2019).* IMCS: International Membrane Computing Society. 2019, pp. 513–522.

[10] G. Rozenberg and A. Salomaa, eds. *Handbook of Formal Languages. 3 vols.* Berlin, Heidelberg: Springer-Verlag, 1997.

[11] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing.* USA: Oxford University Press, Inc., 2010. ISBN: 0199556679.

[12] M. Pérez-Jiménez, Á. Jiménez, and F. Caparrini. “Complexity classes in cellular computing with membranes”. In: *Natural Computing* 2 (Sept. 2003), pp. 265–285. DOI: 10.1023/A:1025449224520.

[13] M. J. Pérez-Jiménez, A. R. Jiménez, and F. S. Caparrini. “Decision P Systems and the P≠NP Conjecture”. In: *Membrane Computing.* Ed. by G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 388–399. ISBN: 978-3-540-36490-0.

[14] A. Cordón-Franco, M. Ángel-Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. “Weak Metrics on Configurations of a P System”. In: *Proceedings of the Second Brainstorming Week on Membrane Computing (2004).* 2004, pp. 139–151.

[15] G. Păun. “Further Twenty Six Open Problems in Membrane Computing”. In: *Proceedings of the Third Brainstorming Week on Membrane Computing.* Ed. by M. Gutiérrez-Naranjo, A. Riscos-Nuñez, F. Romero-Campero, and D. Sburlan. Sevilla: Fénix Editora, 2005, pp. 249–262.

[16] I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez, M. Á. Gutiérrez-Naranjo, and M. Rius-Font. “On a partial affirmative answer for a Păun’s conjecture”. In: *International Journal of Foundations of Computer Science* 22.01 (2011), pp. 55–64. DOI: 10.1142/S0129054111007824. eprint: https://doi.org/10.1142/S0129054111007824.

Polymorphic P Systems with Limited Depth

Anna Kuczik and György Vaszil

Faculty of Informatics, University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
kuczik.anna@inf.unideb.hu
vaszil.gyorgy@inf.unideb.hu

Summary. We investigate the computational power of non-cooperating polymorphic P systems with no additional ingredients and having a membrane structure of limited depth. We show that any ET0L language can be generated by such systems with a membrnae structure of depth three.

Key words: Program as data, P systems with dynamic rules, Polymorphic P systems, P systems with non-cooperating rules, P systems with limited depth

1 Introduction

Polymorphic P systems were introduced in [1] motivated by the idea that the program of a computing device could be viewed as data, therefore, it could also be changed during the course of the computation. In these types of P systems, rules are not statically defined, but are dynamically inferred from the contents of pairs of membranes: The contents of one member of the pair defines the multiset representing the left-hand side of the rule, the contents of the other member defines the right-hand side. As the membranes can contain further membranes, the contents of the pairs, and this way the left- and right-hand sides of rules may change dynamically during the computation.

The initial results presented in [1] show the power of the model. With cooperating rules (rules with left-hand sides with more than one objects) any recursively enumerable set of numbers can be generated, but non-cooperating systems (systems with rules with just one object on the left-hand side) can also generate several interesting languages, mainly based on the fact that an exponential, even super-exponential growth of the number of objects inside the system can be produced.

The study of non-cooperating variants of the model was continued further in [3] with considering the case of "no ingredients", that is, when no special features (not even target indicators) are added to the system. The equivalence of so called strong and weak polymorphism was shown, left polymorhism, right polymorphism, and general polymorphism was defined. As its main contribution, [3] presented a hierarchy of computational power based on the depth of the membrane structure, but the computational power of the non-cooperating variant remained unclear.

In the present work, we intend to take some initial steps in this direction by showing that any ET0L language can be generated using non-cooperating polymorphic P systems (with no other ingredients) of depth three. In the following we first review the necessary definitions, then present an example wehere a simple ET0L system is simulated, then finally generalize the idea of the simulation to a method for generating any ET0L language.

2 Preliminaries

In the following we breiefly define the basic notions we will use. See [6] for more on formal language theory, and [4, 5] for details about membrane computing.

Multisets are sets with multiplicites associated with their elements. Let U be a set. A *multiset* over U is a mapping $M : U \to \mathbb{N}$, $M(a)$, for all $a \in U$, is the multiplicity of a in the multiset M. We can also use the form $(a, M(a))$. If U is finite, $U = \{a_1, a_2, \ldots a_n\}$, then $\{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ can also be represented by a string $w = a_1^{M(a_1)}, a_2^{M(a_2)}, \ldots, a_n^{M(a_n)}$ (and all permutations of this string).

In formal language theory, an *alphabet* V is a finite non-empty set of symbols, its cardinality is denoted by $|V|$. A string generated by V under the operation of concatenation is denoted by V^*, and $V^+ = V^* \setminus \{\lambda\}$ where λ denotes the empty string.

Lindenmayer systems (or *L systems*) are parallel rewriting systems introduced in 1968 by A. Lindenmayer. Several variants of L systems have been developed since then, among these, we will use ET0L systems and languages.

A finite substitution τ over an alphabet V is a function mapping each symbol $a \in V$ into a non-empty finite language over: $V : \tau(a) \subseteq V^*$. We extend τ to words by $\tau(\lambda) = \{\lambda\}$, $\tau(w) = \tau(a_1)\tau(a_2)\ldots\tau(a_n)$ for $w = a_1a_2\ldots a_n$, and to languages by $\tau(L) = \{\tau(w) | w \in L\}$.

An *ET0L system* is a 4-tuple $G = (V, T, U, w)$ where V is an alphabet and $T \subseteq V$ is a terminal alphabet are finite sets, $w \in V^+$ is the initial word of G, and U is a finite set of finite substitutions over V (called the *tables* of U). In a computational step in G, all the symbols of the current sentential form (starting with the axiom) are substituted (or rewritten) using one of the tables of U. The language generated by G consists all terminal strings which can be generated in a series of computational steps (a derivation) from the initial word, that is, $L(G) = \{u \in T^* \mid w \Rightarrow^* u\}$ where $\Rightarrow$ denotes a comptutaional step, and $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$. The family of languages generated by ET0L systems is denoted by $\mathcal{L}(ET0L)$.

It is known (see [2]) that for each ET0L system with an arbitrary number of tables, there exists an ET0L system with only two tables generating the very same language. This means that every task which can be solved by an arbitrary ET0L system can also be solved by a system using two tables. Therefore, in the following example and model, we will assume that ET0L systems have two tables.

Moreover, since we are going to relate ET0L languages to the multiset langugaes of P systems, the most important thing is not the string generated by the ET0L system, but the multiplicities of different letters in the generated strings. We will denote by $N(G)$ the language of multisets corresponding the strings of $L(G)$, and by $\mathcal{L}(NET0L)$ the obtained class of multiset langugaes.

A membrane systms (or *P system*) is a tuple

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

where O is an alphabet of objects, $T \subseteq O$ is the set of terminal objects, μ is the membrane structure, w_i are the multisets giving the initial contents of each membrane $1 \leq i \leq n$, R_i is a finite set of rules for each membrane $1 \leq i \leq n$, and h_o is the label of the output membranes, $h_o \in \{1, \ldots, n\}$.

The membrane structure μ is usually denoted by a string of matching parentheses labelled by the numbers $\{1, \ldots, n\}$, but it can also be represented by a tree with its root labelled by the label of the outermost membrane, and the descendant nodes of each node labelled by the labels of membranes enclosed by the region corresponding to the given node. In the following, the number of nodes encountered during the traversal of the longest path from the root to a leaf in such a tree representation will be called the *depth* of the membrane system. (For example, the membrane system which only has one membrane is of depth 1, the system with two nested membrane is of depth 2.)

The rules in $R_i,\ 1 \leq i \leq n$, are given as multiset rewriting rules, of the form $u \to v$, where $u, v \in V^*$ are strings (understood as representations of multisets). If in such rules, the number of objects in u (the multiset on the left side of the rules) is greater than one, then we say that Π is a system with *cooperation*. Otherwise, it is a $non-cooperative$ system.

The rules in a given R_i are applied in the region enclosed by membrane i in a maximally parallel way, that is, as many rules have to be applied in parallel as possible with the restriction that each object can be rewritten by at most one rule.

2.1 The polymorphic P system model

In polymorphic membrane systems, unlike traditional membrane systems, the rules are not directly defined. The rules are represented by the membranes. The left and right sides of each rule are contained by a membrane. Consequently, the structure of the polymorphic membrane system will be different.

A polymorphic P system is a tuple

$$\Pi = (O, T, \mu, w_s, \langle w_{1L}, w_{1R} \rangle, \ldots, \langle w_{nL}, w_{nR} \rangle, h_o),$$

where O is the alphabet of objects, $T \subseteq O$ is the set of terminal objects, μ is the membrane structure consisting $2n + 1$ membranes labelled by $s, 1L, 1R, \ldots, nL,$

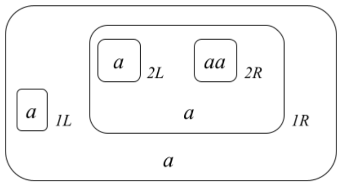

Fig. 1. The polymorphic P system Π_1 of Example 1.

nR, the multiset w_s is the initial contents of the skin membrane, $\langle w_{iL}, w_{iR}\rangle$ are pairs of multisets giving the contents of membranes iL and iR, $1 \leq i \leq n$, and $h_o \in \{1, \ldots, n\}$ is the label of the output membrane.

The *depth* is defined in the same way as conventional membrane systems, it is the height of μ seen as a tree. For every $1 \leq i \leq n$, the membranes iL and iR have the same parent membrane, so they are located at the same depth.

The rules of Π are not given statically in the description, but are dynamically deduced for each configuration based on the content of the membrane pairs iL and iR. Thus, if in the configuration of the system these membranes contain the u and v multisets, then in the next step their parent membrane is transformed as if the $u \to v$ multiset transcription rule were added to it. If iL is empty in some configuration, then the rule defined by the pair iL, iR is considered disabled, that is, no rule will be inferred from the contents of iL and iR.

Similarly to [1], polymorphic membrane systems and their languages are denoted as $NOP^k(polym, ncoo)$ and $\mathcal{L}(NOP^k(polym, ncoo))$ where k denotes the depth, *polym* means polymorphism, and *ncoo* means that the system is $non-cooperative$.

Now we recall an example of a simple polymorphic membrane system with superexponential growth from [1].

Example 1. Consider the polymorphic P system

$$\Pi_1 = (\{a\}, \{a\}, \mu, a, \langle a, a\rangle, \langle a, aa\rangle, S)$$

having a membrane structure as illustrated in Figure 2.1.

In the initial configuration, rule 1 looks like $a \to a$, because of the contents of $1L$ and $1R$, while rule 2 in membrane $1R$ is $a \to aa$. In the first step, rule 1 is applied in the skin, leaving the contents of the membrane intact, and rule 2 is applied in membrane $1R$ doubling the number of a's in 1R, so rule 1 will be changed to the form $a \to aa$. In the second step, rule 1 will transform the multiset a in the skin into aa. and rule 2 is applied in membrane 1R and double the contents again, so after that, rule 1 looks like $a \to a^4$. In general, after k derivation steps, 1R will be a^{2^k}, so rule 1 will have the form $a \to a^{2^k}$. As the number of a's in the skin will be $2^{\frac{k(k-1)}{2}}$, the rate of growth of the contents of the skin membrane is superexponential.

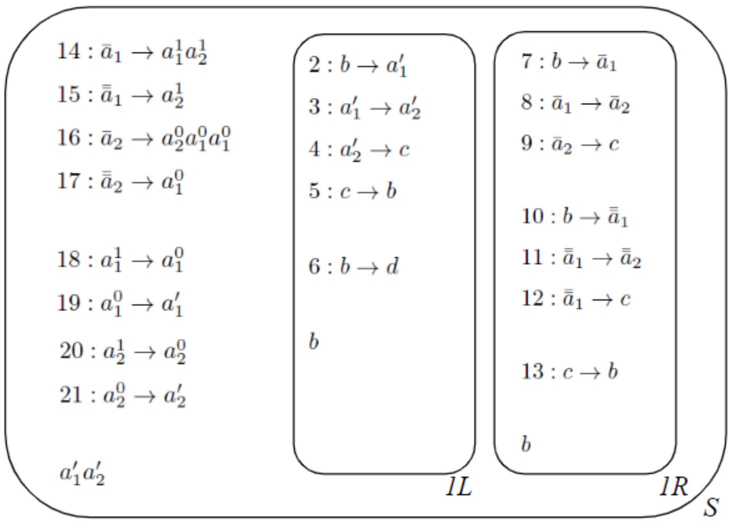
$14 : \bar{a}_1 \to a_1^1 a_2^1$
$15 : \bar{\bar{a}}_1 \to a_2^1$
$16 : \bar{a}_2 \to a_2^0 a_1^0 a_1^0$
$17 : \bar{\bar{a}}_2 \to a_1^0$
$18 : a_1^1 \to a_1^0$
$19 : a_1^0 \to a_1'$
$20 : a_2^1 \to a_2^0$
$21 : a_2^0 \to a_2'$
$a_1' a_2'$
$2 : b \to a_1'$
$3 : a_1' \to a_2'$
$4 : a_2' \to c$
$5 : c \to b$
$6 : b \to d$
b
$1L$
$7 : b \to \bar{a}_1$
$8 : \bar{a}_1 \to \bar{a}_2$
$9 : \bar{a}_2 \to c$
$10 : b \to \bar{\bar{a}}_1$
$11 : \bar{\bar{a}}_1 \to \bar{\bar{a}}_2$
$12 : \bar{\bar{a}}_1 \to c$
$13 : c \to b$
b
$1R$
S

Fig. 2. The P system Π of Example 2.

3 Polymorphic P systems with limited depth

In this section we would like to examine the relationship of languages generated by ET0L systems and simple polymorphic P systems, where simplicity is captured by non-cooperation and limited depth. We look at an example first.

Example 2. Consider the following ET0L system $G = (V, T, U, w)$ with $V = T = \{a_1, a_2\}$, $w = a_1 a_2$, and two tables $U = (P_1, P_2)$, each containing two rules

$$P_1 = \{a_1 \to a_1 a_2,\ a_2 \to a_2 a_1 a_1\}, \text{ and}$$
$$P_2 = \{a_1 \to a_2,\ a_2 \to a_1\}.$$

We construct a non-cooperative polymorphic P system Π with depth 3 that can perform the choosing between rules of P_1 and P_2, and therefore simulates the operation of G.

Let $O = \{a_1, a_2, a_1', a_2', a_1^0, a_2^0, a_1^1, a_2^1, \bar{a}_1, \bar{a}_2, \bar{\bar{a}}_1, \bar{\bar{a}}_2, b, c, d, \}$, $T' = \{a_1', a_2'\}$ and

$$\Pi_2 = (O, T', \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \dots, \langle w_{21L}, w_{21R}\rangle, s)$$

where the membrane structure of Π is defined as

$$\mu = [\ [\dots]_{1L}\ [\dots]_{1R}\ [\ ]_{14L}\ [\ ]_{14R}\ \dots [\ ]_{21L}\ [\ ]_{21R}\ ]_s$$

with membrane $1L$ containing the inner membranes $[\ ]_{2L}\ [\ ]_{2R} \dots [\ ]_{6L}\ [\ ]_{6R}$, and membrane $1R$ containing the inner membranes $[\ ]_{7L}\ [\ ]_{7R} \dots [\ ]_{13L}\ [\ ]_{13R}$.

We use two types of rules in polymorphic membrane systems. The rules belonging to the first type do not change during the solution of the task, while the rules belonging to the second type can change step by step. In this example we only have one rule that changes during the steps, rule 1. The other rules have the same form at every step during the process.

The graphical representation of μ can be seen in Figure 2 where also the initial membrane contents are depicted. Non-dynamical rules, that is, pairs of membranes $[\ w_{iL}\ ]_{iL}$, $[\ w_{iR}\ ]_{iR}$ with constant contents (contents that never change during the computation) are given in a simplified notation as $w_{iL} \to w_{iR}$.

The initial contents of the regions effected by the polymorphic nature of Π, that is, the regions with non-constant contents are

$$w_s = a_1' a_2',\ w_{1L} = b,\ w_{1R} = b.$$

Step	Rule 1	Skin	1L	1R	Rules 14-21
1.	$b \to b$	$a_1' a_2'$	2	7	
2.	$a_1' \to \bar{a}_1$	$a_1' a_2'$	3	8	
3.	$a_2' \to \bar{a}_2$	$\bar{a}_1 a_2'$	4	9	14
4.	$c \to c$	$a_1^1 a_2^1 \bar{a}_2$	5	13	$16, 18, 20$
5.	$b \to b$	$a_1^0 a_2^0 a_2^0 a_1^0 a_1^0$	2	7 or 10	$19, 21$
6.	$a_1' \to \bar{a}_1$ or $a_1' \to \bar{\bar{a}}_1$	$a_1' a_2' a_2' a_1' a_1'$	...	...	...

Table 1. The polymorphic system Π of example 2

The functioning of Π is demonstrated in Table 1. The first column contains the step number. The second column contains how rule 1, defined by the membranes $1L$ and $1R$ looks like after every step. The third column contains the elements of the skin region. The fourth, fifth, and sixth columns contain the rules we (need to) use in the corresponding steps.

The general idea behind the functioning of Π_2 is as follows. Rules $14 - 17$ simulate the rewriting process of the tables of G. Those with lefthand side $\bar{a}_1$ or $\bar{a}_2$ simulate the first table, those with lefthand side $\bar{\bar{a}}_1, \bar{\bar{b}}_2$ simulate the second table. The objects of the skin region correspond to the sentential form of G. Rule 1 is "dynamic", it prepares the objects of the skin membrane for the application of the rules $14 - 17$ in the appropriate order. At the beginning of a "simulating cycle", rule 1 is used to rewrite a (more precisely, its variant, a_1') to $\bar{a}_1$ or $\bar{\bar{a}}_2$, selecting this way the table to be simulated. Then, rule 1 changes to rewrite a_2' according to the same selection, while rules $14 - 17$ proceed with the actual simulation of the chosen table. The rest of the rules are needed to synchronize the whole process.

Table 1 shows how the rewriting of $a_1 a_2$ to $a_1 a_2 a_2 a_1 a_1$ by the first table of G is simulated in Π. In the initial state, rule 1 looks like $b \to b$, which is not applicable, because we only have an a_1' and a a_2' in the skin.

So in the first step, we have to change rule 1. In $1L$ we can use rule 2 ($b \to a_1'$), which rewrites b in $1L$ to a_1' making rule 1 is applicable, because we can use it to write a_1' in the skin. In parallel, we have to use rule 7 ($b \to \bar{a}_1$) or rule 10 ($b \to \bar{\bar{a}}_2$) in $1R$. This decision depends on which table of the ET0L system we want to simulate. To simulate P_1, we must use rule 7, and to simulate P_2, we must use rule 10. As we would like to simulate P_1, we use rule 7.

As can be seen in the second row of Table 1, the form of rule 1 has changed, and now we can use it in the skin and rewrite a_1' to $\bar{a}_1$.

At the same time, the rules used in $1L$ and $1R$ ($a_1' \to a_2'$, $\bar{a}_1 \to \bar{\bar{a}}_2$, respectively), changed the shape of rule 1 to $a_2' \to \bar{\bar{a}}_2$, in order to be able to start rewriting a_2'-s in the next step.

After we used rule 1, the object(s) in the skin changed. Now, we can use rule 14 ($\bar{a}_1 \to a_1^1 a_2^1$), which simulates the first rule from the P_1 ET0L table of G. The upper indexing of the symbols on the right-hand side starts from 1, so that they are written back into the primed form (after counting down with the indices to zero) at the appropriate step.

Meanwhile, in step 3, rule 1 ($a_2' \to \bar{\bar{a}}_2$) is also applied to rewrite a_2' (so the second rule of table P_1 of G can also be simulated), and rule 1 is changed to $c \to c$ (so it cannot be applied in the next step).

Now, with rule 16 ($\bar{\bar{a}}_2 \to a_2^0 a_1^0 a_1^0$), the rule $a_2 \to a_2 a_1 a_1$ of G is simulated, while rules 18 and 20 decrement the upper indices of the objects introduced by the simulation of the previous rule, and rule 1 is changed to $b \to b$.

Now, as can be seen in row 5. of Table 1, the system is ready to prepare the next simulating cycle by rewriting the objects corresponding to the sentential form of G to their primed versions, and changing rule 1 in the appropriate way. We returned to a state that was similar to the initial state, where $1L$ has b and $1R$ also has b, so we can choose between rule 7 and rule 10 again (to simulate another step from the ET0L system), and in parallel, rewrite a_1^0-s and a_2^0-s to a_1'-s and a_2'-s, with rules 19 and 21.

The simulation of the ET0L system can be completed when it returns to a state that is similar to the initial state, so before another table is selected for simulation. So, if $1L$ has b and $1R$ also has b, and we want to stop the mechanism, then we can choose rule 6 instead of rule 7 or rule 10. Rule 6 shuts down the system and the simulation ends. The reason for this is that after applying rule 6, the form of rule 1 is: $d \to \bar{a}_1$ or $d \to \bar{\bar{a}}_1$, and we can't use any of these rules in the Skin membrane, because we don't have a d object in Skin.

The result will be a string made from the letters $\{a_1', a_2', \ldots\}$ in the *Skin* membrane of the system.

Now we show that the basic idea of the example above can be generalized to arbitrary ET0L systems. Without the loss of generality, we assume that we deal with systems having two tables.

Theorem 1. $\mathcal{L}(NET0L) \subseteq \mathcal{L}(NOP^3(polym, ncoo))$.

Proof. Let $G = (N, T, U, w)$ be an ET0L system and let k denote the number of letters in the alphabet, $N = \{a_1, a_2, ..., a_k\}$. The two tables are $U = (P_1, P_2)$, and the rules are denoted by $r_{i,j}$, where index $i \in \{1, 2\}$ denotes the index of the table, and $1 \leq j \leq m$ is the index of the rule, with m being the maximum of $|P_1|$ and $|P_2|$, the number of rules in the two tables.

We denote the left- and righthand sides of the rules as $r_{i,j} : \alpha_{i,j} \to \beta_{i,j}$, where $\alpha_{i,j} \in N$ and $\beta_{i,j} \in N^*$, $1 \leq i \leq 2,\ 1 \leq j \leq m$.

Let

$$O = \{a_i, a'_i, a^n_i, \bar{a}_i, \bar{\bar{a}}_i \mid 1 \leq i, n \leq k\} \cup \{b, c, d, \},\ T' = \{a'_i \mid 1 \leq i \leq k\}$$

and let

$$\Pi = (O, T', \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \ldots, \langle w_{pL}, w_{pR}\rangle, s)$$

where $p = 2 + (3k + 6) + 2m + \frac{k(k+1)}{2}$, and the membrane structure of Π is defined as

$$\mu = [\ [\ldots]_{1L}\ [\ldots]_{1R}\ [\ ]_{(3k+7)L}\ [\ ]_{(3k+7)R}\ \cdots [\ ]_{pL}\ [\ ]_{pR}\ ]_s$$

with membrane $1L$ containing the inner membranes $[\ ]_{2L}\ [\ ]_{2R} \cdots [\ ]_{6L}\ [\ ]_{6R}$, and membrane $1R$ containing the inner membranes $[\ ]_{7L}\ [\ ]_{7R} \cdots [\ ]_{13L}\ [\ ]_{13R}$.

In 1L, the number of rules depends on the number of letters in the alphabet of the ET0L system, we have to apply $k + 2$ rules for each table simulation in succession, where $k = |N|$. In general, we specify the rules for k letters as

$$b \to a'_1,\ a'_i \to a'_{i+1},\ \text{for } 1 \leq i \leq k - 1,\ \text{and } a'_k \to c,\ c \to b.$$

They perform the same task as the rules of 1L in Example 2 do for two letters.

Note that here we have used the simplified notation again for membranes with contents that remain constant for the whole computation. (Without this simplification we would have to write $\langle w_{2L}, w_{2R}\rangle$ and specify $w_{2L} = b$, $w_{2R} = a'_1$ instead of the rule $b \to a'_1$, and so on.)

Rules must be applied in $1R$ depending on the choice of the table, because those rules give the right side of rule 1, which modifies the objects in the skin. So we have to create rules for also P_1 and P_2, like in the example. We need

$$b \to \bar{a}_1,\ \bar{a}_i \to \bar{a}_{i+1} \text{ for } 1 \leq i \leq k - 1,\ \bar{a}_k \to c,\ c \to b,$$

and

$$b \to \bar{\bar{a}}_1,\ \bar{\bar{a}}_i \to \bar{\bar{a}}_{i+1} \text{ for } 1 \leq i \leq k - 1,\ \bar{\bar{a}}_k \to c,\ c \to b.$$

In the skin region we have to go through the rules of a table in order, and for this reason we do not rewrite the letters at the same time. We have to add extra rules, which help us get back to the form $a'_1, a'_2, \ldots$ for all objects at the same step, after rewriting the last letter.

We denote the jth rule of table 1 and table 2 with $r_{1,j}$ and $r_{2,j}$, respectively. In order to simplify the notation, we assume that the cardinality of the two tables are the same, $m = |P_1| = |P_2|$. If this is not the case, $|P_1| < |P_2|$ for example, then we consider the "missing rules" $r_{1,j},\ P_1| < j \leq |P_2|$, to be $a_1 \to a_1$. Now we add the following rules to the skin region.

Rule set for the rules of P_1, the first table of $U = (P_1, P_2)$:

$$\{\bar{a}_i \to \beta_{1,j}^{k+1-i} \mid \alpha_{1,j} \to \beta_{1,j} \in P_1, \alpha_{1,j} = a_i \text{ for some } a_i \in \{a_1, a_2, \dots a_k\}\}.$$

Rule set for the rules of P_2 of $U = (P_1, P_2)$:

$$\{\bar{\bar{a}}_i \to \beta_{2,j}^{k+1-i} \mid \alpha_{2,j} \to \beta_{2,j} \in P2, \alpha_{2,j} = a_i \text{ for some } a_i \in \{a_1, a_2, \dots a_k\}\}.$$

After rewriting with the rules above, we have to use rules to count down a_i-s indexes until the last letter is transcribed, similarly as we count down in the example. Thus, every a_i gets $k + 1 - i$ indices as follows.

$$a_i^n \to a_i^{n-1}, a_i^1 \to a_i^{'}, \text{ where } 2 \leq n \leq k + 1 - i, 1 \leq i \leq k - 1.$$

To stop the system, we need to add another rule to $1L$. The rule must be one that can be applied at the end of a table simulation. For this reason, when $1L$ and $1R$ return to a state similar to the initial state, there should be an option to apply the stopping rule, which is $b \to d$, similar to the system of Example 2.

So we add the rule

$$b \to d$$

to $1L$.

After we used this rule the system shut down, because we never have d object in the skin region. So after this step we can't continue the rule application, the system halts. The result will be a string consisting of the letters $\{a_1', a_2', \dots\}$ in the skin membrane of the system.

4 Conclusion

We have shown how a simple ET0L system can be simulated by a non-cooperating polymorphic P system of depth three, and then generalized the idea to produce any ET0L language. Our work is intended to be the initial step in the investigation of the computing power of non-cooperating systems with limited depth. The next topic of further study is looking for upper bounds on the computational power, and in particular, to relate left and right polymorphism and their depth-limited variants to the general case.

References

1. Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic P systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 81–94, Berlin, Heidelberg, 2011. Springer-Verlag.

2. Andrzej Ehrenfeucht, Grzegorz Rozenberg, and Sven Skyum. A relationship between ET0L and EDT0L languages. *Theoretical Computer Science*, 1(4):325–330, 1976.
3. Sergiu Ivanov. Polymorphic P systems with non-cooperative rules and no ingredients. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing*, volume 8961 of *Lecture Notes in Computer Science*, pages 258–273, Cham, 2014. Springer International Publishing.
4. Gheorghe Păun. *Membrane Computing: An Introduction.* Springer-Verlag, Berlin, Heidelberg, 2002.
5. Gheorghe Păun, Grzegorz Rozenberg, and Aarto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, Oxford, 2010.
6. Grzegorz Rozenberg and Aarto Salomaa, editors. *Handbook of Formal Languages.* Springer-Verlag, Berlin Heidelberg, 1997.

Security analysis of a key agreement protocol based on Spiking Neural P Systems

Mihail-Iulian Pleșa*[1], Marian Gheoghe[2], Florentin Ipate[1], and Gexiang Zhang[3]

[1] Department of Computer Science, University of Bucharest, Bucharest, Romania
`mihail-iulian.plesa@s.unibuc.ro`

[2] School of Electrical Engineering and Computer Science, University of Bradford, Bradford, UK

[3] School of Automation, Chengdu University of Information Technology, Chengdu 610225, China

Summary. Key agreement protocols are a central part of cryptography research. The idea of such a protocol is to allow two or more parties to reach a shared secret by communicating over a public channel. There are three main types of key agreement protocols: based on hard mathematical problems, based on quantum effects and based on neural synchronization. Although they have not been much studied until now, key agreement protocols based on neural synchronization have several advantages. First, their security does not involve any number theory problem which may be solved on a quantum computer. Second, unlike quantum key agreement protocols, they do not require special hardware which is expensive and hard to manage. Recently, a new neural key agreement protocol based on the Anti-Spiking Neural Tree Parity Machine P system, ASNTPM P system for short, has been proposed. Although the protocol is more efficient than the rest of the neural key agreement protocols, no security analysis was performed.

In this paper, we study the security of the protocol based on ASNTPM P systems from a cryptographic perspective. We analyze the running time of the protocol with respect to the parameters of the system. We adopt multiple attacks from the neural cryptography literature and show that the ASNTPM P system-based protocol is secure. Through a series of experiments, we show that the running time of the protocol grows polynomially in the system parameters while the probability that an attack will succeed decreases exponentially.

Key words: Spiking Neural P system, Anti-spikes, ASNTPM P systems, Tree Parity Machine, Cryptography

1 Introduction

Key agreement protocols are the basis of all modern cryptographic protocols. From simple web traffic which is secured using the TLS protocol to the more complex end-to-end encrypted messaging communication platforms which are based on Signal protocol, most cryptographic systems use some sort of key agreement protocols [10, 11, 42]. The main role of such protocols is to establish a shared secret among two or more parties using public communication channels.

Currently, most key agreement protocols are based on hard number theory problems e.g., the discrete logarithm problem (DLP), the Diffie-Hellman problem (DHP), the decisional Diffie-Hellman problem (D-DHP), the factorization problem, etc [15]. The disadvantage of these protocols is that they are vulnerable if a large quantum computing is built. In [47], Peter Shor proposed a quantum algorithm that can solve the DLP and the factorization problem in polynomial time. Until this moment, there is no large enough quantum computer to endanger the cryptogamic primitives and protocols deployed in the industry. However, it is expected that such a computer will be built shortly [38].

There are three alternatives to the current key agreement protocols:

1. Post-quantum key agreement protocols
2. Quantum key agreement protocols
3. Neural key agreement protocols

The post-quantum key agreement protocols are based on hard mathematical problems for which no efficient solution is known on classical or quantum computers [4]. The disadvantage is that there is no mathematical proof that there is indeed no solution to those problems. The quantum key agreement protocols are based on quantum effects e.g., the collapse of the probability wave, entanglement, no-cloning theorem, etc. [23]. Although these protocols are secure even if a large quantum computer is built, they require specialized hardware which is hard to manage and expensive.

There are many important applications of neural networks [1,2,16,25,27,29–32, 36,39,41]. A less known application is the construction of key agreement protocols. The neural key agreement protocols represent an alternative to post-quantum or quantum protocols. The idea behind these protocols is to synchronize over a public channel two special neural networks called Tree Parity Machines, TPMs for short [21]. Unlike post-quantum key agreement protocols, they are not based on any hard mathematical problem so they are quantum secure. Their principal advantage over the quantum key agreement protocols is that they do not require special hardware and can be implemented on any classical computer.

Until recently, most TPMs were constructed using neurons modeled after the perceptrons. In [40], the authors proposed for the first time a neural key agreement protocol based on Spiking Neural P systems. They constructed a special type of TPM called the Anti-Spiking Neural Tree Parity P system (ASNTPM P system) and showed experimentally that their protocol is more efficient than the classical neural key agreement protocols based on TPMs. In this paper, we study the security of this protocol from a cryptographic perspective. We use the most simple security model in which the attacker only eavesdrops on the communication channel. The attacker is not allowed to alter the messages exchanged by the legitimate parties nor to insert or delete messages. We adopt multiple attacks on neural key agreement protocols from the literature and test whether the ASNTPM P system-based protocol is secure against them. The paper is organized as follows: in Section 2 we present related work and our contribution. In Section 3 we introduce the definition of the model proposed by [40]. In Section 4 we analyze the running time of the protocol with respect to the parameters of the system. In Section 5 we discuss four different types of attacks against neural cryptography and show experimentally that the protocol proposed in [40] is secure. Section 5 is left for conclusions.

2 Related work

The idea of using neural synchronization to build a key agreement protocol was first proposed in [21]. The TPM proposed by the authors was a three-layer neural network with binary inputs. Shortly after the idea was launched, three types of attacks were proposed in [22]. The authors showed through multiple experiments that they can recover more the 90% of the shared key using the geometric attack. In [28] the authors experimentally proved that increasing the range of the weight can improve the security of the protocol. The paper provides evidence that increasing the range increases the synchronization time polynomially while decreasing the probability that the geometric attack will succeed exponentially. In [46] is presented a more powerful attack than the geometric one. This attack called the majority attack cannot be mitigated by increasing the range of the weights.

There are also other strategies for improving the security of neural key agreement protocols. In [43] the authors proposed a mechanism by which the inputs of the TPM are generated based on the current internal state of the network. In [7] and [8] the authors presented two algorithms for perturbing the output of the TPM in such a way that the attacker cannot recover the original information but the two legitimate parties can synchronize. This improves the security of the protocol because every attack uses the fact that the eavesdropper can intercept the outputs exchanged by legitimate parties over a public channel.

In [51] and [20] the authors proposed other architectures for constructing a TPM. In [51] the idea of using non-binary input values is presented improving the running time of the protocol. In [20] it is shown that using vector values as inputs can further improve the efficiency and the security of the protocol. Similarly, [12] proposed a TPM with complex input values. In [52] the authors analyzed the impact of non-binary input values on the security of a TPM-based key agreement protocol. Regarding the practical aspects of neural cryptography, in [45] several sets of parameters for TPMs were analyzed. For each set, the authors presented the synchronization time and the security impact.

In this work, we are dealing with TPM instantiated with Spiking Neural P systems. These systems are a special type of the membrane computing model introduced in [37]. Membrane computing models have been used to solve hard problems like Hamiltonian Path in polynomial time [58]. In [6] the authors proposed a new sorting algorithm based on P systems. There are also P systems inspired by various physical phenomena. In [5] the authors presented a P system in which the membranes have a limited capacity and [17] presents a new model inspired by the controlled circulation of water. Also, there are attempts to make these models in the laboratory [26].

Spiking neural P systems were first introduced in [18]. Over time, new functionalities inspired by various biological phenomena were added to the original model. The most known Spiking Neural P systems are the following [9, 33–35, 49, 56, 57]:

1. SN P systems with astrocytes
2. SN P systems with communication on request
3. SN P systems with polarizations
4. SN P systems with colored spikes
5. SN P systems with asynchronous systems
6. SN P systems with anti-spikes
7. SN P systems with a flat maximally parallel use of rules

The protocol analyzed in this paper is instantiated with a TPM based on Spiking Neural P system with anti-spikes. Several variations of this model include Spiking Neural P systems without the annihilating priority [55] and Spiking Neural P systems with multiple channels [50]. Spiking Neural P systems were studied from both a practical and a theoretical perspective. The computational power of SN P systems with multiple channels was investigated in [24] while a formal verification of SN P systems by mapping them to kernel P systems was made in [14, 19]. SN P systems were also used to simulate uniform sequential computing models [3].

Apart from key agreement protocol, SN P systems have other applications in cryptography [48, 59]. In [13] the authors implemented the famous RSA algorithm using SN P systems and in [54] the authors used a variant of SN P systems to break the same cryptosystem by proposing a new and efficient factorization procedure [53].

2.1 Our contribution

In this paper, we make a security analysis of the key agreement protocol proposed in [40]. In the paper, the authors proposed a new TPM called Anti-Spiking Neural Tree Parity Mchine which is based on SN P systems with anti-spikes. We propose a new algorithm for computing the synchronization percentage between two AS-NTPM P systems and also study the efficiency of the protocol with respect to the parameters of the SN P system. Our main contribution is a series of experiments in which we show that increasing the number of input neurons or the number of hidden neurons can exponentially decrease the percentage of the key recovered by the attacker. This growth in the number of neurons increases the synchronization time only polynomially. Our experiments adopt known attacks on TPM-based key agreement protocols in a simple security model in which the attacker can only eavesdrop on the messages exchanged by the legitimate parties.

3 ASNTPM P Systems

The definition of an ASNTPM P system as stated in [40] is the following:

Definition 1. *An Anti-Spiking Neural Tree Parity Machine P system is defined as the following construct:*

$$\Pi = (O, \{\sigma_{in_{11}}, \sigma_{in_{12}}, ..., \sigma_{in_{KN}}\}, \{\sigma_{h_1}, \sigma_{h_2}, ..., \sigma_{h_K}\}, \sigma_{out}, N, K, L, syn_0, f)$$

where:

1. $O = \{a, \overline{a}\}$ is an alphabet formed by two symbols:
   a) The symbol a denotes a spike
   b) The symbol $\overline{a}$ denotes an anti-spike
2. $\sigma_{in_{ij}}$ is an input neuron formed by the following tuple $(n_{ij}, \overline{n}_{ij}, R_{ij})$:
   a) n_{ij} denotes the number of spikes from the neuron
   b) $\overline{n}_{ij}$ denotes the number of anti-spikes from the neuron
   c) R_{ij} is a finite set of rules of the following forms:
      i. Firing rules: $b^c \rightarrow b^{c*w_{ij}}$ where $b \in \{a, \overline{a}\}$, $1 \leq c \leq L$ and w_{ij} is a positive integer that will be defined below.
      If at the moment t a neuron has c spikes or anti-spikes it will fire, consuming either c spikes or c anti-spikes and sending $c * w_{ij}$ spikes or $c * w_{ij}$ anti-spikes to the hidden neuron $\sigma_{h_i}$with which it is connected. At moment $t = 0$, there are no spikes or anti-spikes in the neuron i.e., $n_{ij} = 0$, $\overline{n}_{ij} = 0$.
      ii. Annlihilation rule: $a\overline{a} \rightarrow \lambda$
      This rule indicates that at any moment t, an input neuron cannot contain spikes and anti-spikes simultaneously. If a spike and an anti-spike are present in an input neuron, they will annihilate each other instantaneously.

   Here, $1 \leq i \leq K$ and $1 \leq j \leq N$.
3. σ_{h_i} is a hidden neuron formed by the following tuple $(n_i, \overline{n}_i, R_i)$:
   a) n_i denotes the number of spikes from the neuron
   b) $\overline{n}_i$ denotes the number of anti-spikes from the neuron
   c) R_i is a finite set of rules of the following form:
      i. Firing rules: rule of the form $b^c \rightarrow b$ where $b \in \{a, \overline{a}\}$, $1 \leq c \leq NL$.
      If at the moment t the neuron has c spikes it will fire consuming c spikes and sending 1 spike to the output neuron. If at the moment t the neuron has c anti-spikes it will fire consuming c anti-spikes and sending 1 anti-spike to the output neuron. At moment $t = 0$, there are no spikes or anti-spikes in the neuron i.e., $n_i = 0$, $\overline{n}_i = 0$.
      ii. Annlihilation rule: $a\overline{a} \rightarrow \lambda$
      This rule indicates that at any moment t, a hidden neuron cannot contain spikes and anti-spikes simultaneously. If a spike and an anti-spike are present in a hidden neuron, they will annihilate each other instantaneously.

   Here, $1 \leq i \leq K$.

4. σ_{out} is the output neuron formed by the following tuple $(n_{out}, \overline{n}_{out}, r_{out})$:
   a) n_{out} denotes the number of spikes from the neuron
   b) $\overline{n}_{out}$ denotes the number of anti-spikes from the neuron
   c) r_{out} is an annihilation rule of the form $a\overline{a} \rightarrow \lambda$. The rule indicates that the output neuron cannot hold spikes and anti-spikes simultaneously. At moment $t = 0$, there are no spikes or anti-spikes in the neuron i.e., $n_{out} = 0$, $\overline{n}_{out} = 0$.

   The output of the system is the number of spikes or anti-spikes from this neuron.
5. syn_t is the set of synapses at the computational step t. A synapse is defined by the triplet $(\sigma_i, \sigma_j, w_{ij})$ meaning the existence of a synapse between the neuron σ_i and the neuron σ_j. Here, σ_i and σ_j can be input, hidden, or output neurons. The weight on the synapse, $w_{ij} \in \mathbb{Z}^+$ has the role to amplify the spikes or the anti-spikes passing through the synapse e.g., if the neuron σ_i fires sending c spikes or anti-spikes to the neuron σ_j and the weight on the synapse is w_{ij} then the neuron σ_j will receive $c * w_{ij}$ spikes or anti-spikes.
   syn_0 represents the set of synapses at moment $t = 0$. Initially, the weights between the input and the hidden neurons are randomly chosen from the set $\{1, 2, ..., L\}$. The weights between the hidden and the output neurons are always 1.
6. N is the number of input neurons connected to a single hidden neuron.
7. K is the number of neurons hidden neurons.
8. L represents the maximum value of a weight i.e., $0 < w_{ij} \leq L$.
9. The learning function f has the role of updating the weights on the synapses according to (1):
   $$syn_{t+1} = f(syn_t) \tag{1}$$
   The exact form of the learning function f is described by the procedure from Algorithm 3.

The system is initialized using the initialization procedure described in Algorithm 1. The input consists of the ASNTPM P System and a vector of $N * K$ elements $X = (x_{11}, x_{12}, ..., x_{KN})$, $-L \leq x_{ij} \leq L, x_{ij} \neq 0$, $\forall 1 \leq i \leq K, 1 \leq j \leq N$. The input of the system is defined by the vector. If $x_{ij} < 0$ then the input neuron σ_{ij} will receive from the environment $|x_{ij}|$ anti-spikes. If on the other hand, $x_{ij} > 0$ then the input neuron σ_{ij} will receive from the environment x_{ij} spikes. The initialization procedure is described by Algorithm 1.

The input neurons fire sending all the spikes or all the anti-spikes to the hidden neurons. The number of spikes or anti-spikes is amplified by the corresponding weight of the synapse. After the annihilation rule is applied the maximum number of times in each hidden neuron, they send one spike or one anti-spike to the output neuron. After the annihilation rule is applied the maximum number of times in the output neuron, it can be in one of the following states:

1. The output neuron is empty if the number of spikes received from the hidden neurons is equal to the number of anti-spikes.

2. The output neuron contains one spike if the number of spikes received from the hidden neurons is greater than the number of anti-spikes.
3. The output neuron contains one anti-spike if the number of spikes received from the hidden neurons is smaller than the number of anti-spikes.

The output of the system is the state of the output neuron. The system evolves by the application of the learning function f which modifies the weights of the synapses between the input and the hidden neurons. The running procedure of an ANSTPM P System is described in Algorithm 2. A generic ASNTPM P System is presented in Figure 1.

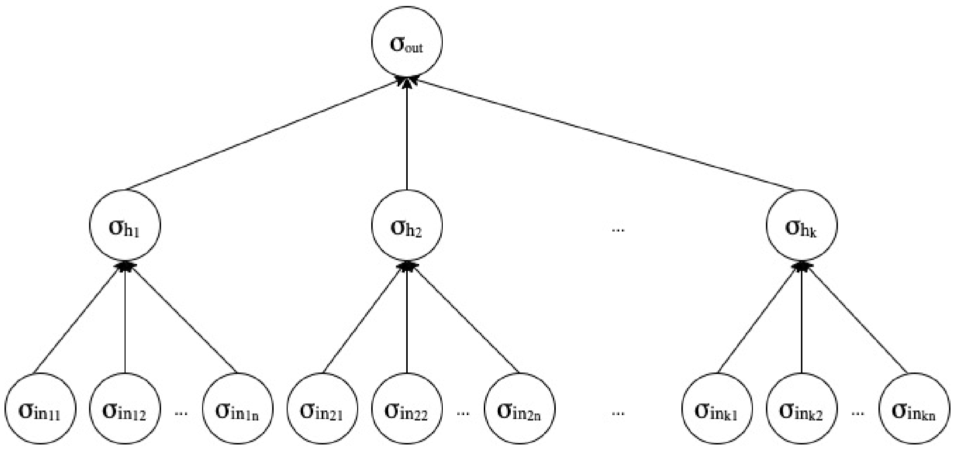

Fig. 1. A generic ASNTPM P System

Let $N(\Pi, \sigma)$ be the number of spikes from neuron σ of the ASNTPM Π. Similarly, let $\overline{N}(\Pi, \sigma)$ be the number of anti-spikes from neuron σ of the ASNTPM Π. Here σ can be an input, a hidden, or an output neuron.

Let W_Π be the weights on the synapses of the ASNTPM Π. More exactly, W_Π is a $K \times N$ matrix in which the element on line i and column j denoted as $W_\Pi[i][j]$ represent the weight between the hidden neuron σ_{h_i} and the input neuron $\sigma_{in_{ij}}$.

Algorithm 1 ASNTPM P System initialization

```
1: **function** INITIALIZE($\mathit{\Pi}, X$)
2:     **for** $i = 1;\ i \leq K;\ i = i + 1$ **do**
3:         **for** $j = 1\ j \leq N\ j = j + 1$ **do**
4:             **if** $X[i * N + j] \leq 0$ **then**
5:                 $\overline{N}(\mathit{\Pi}, \sigma_{in_{ij}}) = |X[i * N + j]|$
6:             **else**
7:                 $N(\mathit{\Pi}, \sigma_{in_{ij}}) = X[i * N + j]$
8:             **end if**
9:             $W_{\mathit{\Pi}}[i][j] \overset{\$}{\leftarrow} [0, 2L]$
10:        **end for**
11:    **end for**
12:    **for** $i = 1;\ i \leq K;\ i = i + 1$ **do**
13:        $\overline{N}(\mathit{\Pi}, \sigma_{h_i}) = 0$
14:        $N(\mathit{\Pi}, \sigma_{h_i}) = 0$
15:    **end for**
16:    $\overline{N}(\mathit{\Pi}, \sigma_{out}) = 0$
17:    $N(\mathit{\Pi}, \sigma_{out}) = 0$
18: **end function**
```

Algorithm 2 ASNTPM P System running

```
1: **function** RUN($\mathit{\Pi}$)
2:     **for** $i = 1;\ i \leq K;\ i = i + 1$ **do**
3:         **for** $j = 1\ j \leq N\ j = j + 1$ **do**
4:             $\overline{N}(\mathit{\Pi}, \sigma_{h_i}) = \overline{N}(\mathit{\Pi}, \sigma_{h_i}) + W_{\mathit{\Pi}}[i][j] * \overline{N}(\mathit{\Pi}, \sigma_{in_{ij}})$
5:             $N(\mathit{\Pi}, \sigma_{h_i}) = N(\mathit{\Pi}, \sigma_{h_i}) + W_{\mathit{\Pi}}[i][j] * N(\mathit{\Pi}, \sigma_{in_{ij}})$
6:         **end for**
7:     **end for**
8:     **for** $i = 1;\ i \leq K;\ i = i + 1$ **do**
9:         $\overline{N}(\mathit{\Pi}, \sigma_{out}) = \overline{N}(\mathit{\Pi}, \sigma_{out}) + \overline{N}(\mathit{\Pi}, \sigma_{h_i})$
10:        $N(\mathit{\Pi}, \sigma_{out}) = N(\mathit{\Pi}, \sigma_{out}) + N(\mathit{\Pi}, \sigma_{h_i})$
11:    **end for**
12: **end function**
```

Algorithm 3 The learning function

```
function UpdateWeights($\Pi$)
    for $i = 1;\ i \leq K;\ i = i + 1$ do
        if $[[N(\Pi, \sigma_{h_i}) = N(\Pi, \sigma_{out})] \vee [\overline{N}(\Pi, \sigma_{h_i}) = \overline{N}(\Pi, \sigma_{out})]]$ then
            for $j = 1\ j \leq N\ j = j + 1$ do
                if $N(\Pi, \sigma_{out}) > 0$ then
                    $W_\Pi[i][j] = \left|W_\Pi[i][j] + N(\Pi, \sigma_{in_{ij}})\right|$
                else if $\overline{N}(\Pi, \sigma_{out}) > 0$ then
                    $W_\Pi[i][j] = \left|W_\Pi[i][j] - \overline{N}(\Pi, \sigma_{in_{ij}})\right|$
                end if
            end for
            if $W_\Pi[i][j] > L$ then
                $W_\Pi[i][j] = L$
            end if
        end if
    end for
end function
```

4 The synchronization of two ASNTPM P Systems

Two ASNTPM P Systems are synchronized if their weights are identical. To formalize this idea, we introduce a new quantitative indicator called the synchronization percentage which describes how much two ASNTPM P Systems are synchronized. This indicator is computed by the function *SynchronizationPercentage* presented in Algorithm 4.

Algorithm 4 The synchronization percentage

```
function SynchronizationPercentage($\Pi_1, \Pi_2$)
    $total \leftarrow 0$
    $counter \leftarrow 0$
    for $i = 1\ i \leq K\ i = i + 1$ do
        for $j = 1\ j \leq N\ j = j + 1$ do
            if $W_{\Pi_1}[i][j] \neq 2L \wedge W_{\Pi_2}[i][j] \neq 2L$ then
                $total \leftarrow total + 1$
                if $W_{\Pi_1}[i][j] = W_{\Pi_2}[i][j]$ then
                    $counter \leftarrow counter + 1$
                end if
            end if
        end for
    end for
    return $counter/total$
end function
```

A simple function for synchronizing two ASNTPM P Systems defined by the same parameters K, N and L is presented in Algorithm 5. Since both systems update their weights based on their mutual output, we say that the two ASNTPM P systems are synchronizing with mutual learning.

Algorithm 5 Synchronization of two ASNTPM P Systems

```
 1: function Synchronize($\Pi_1, \Pi_2$)
 2:     while SynchronizationPercentage($\Pi_1, \Pi_2$) $\neq 1$ do
 3:         $x \overset{R}{\leftarrow} \mathbb{Z}^{KN}$
 4:         Initialize($\Pi_1, x$)
 5:         Initialize($\Pi_2, x$)
 6:         Run($\Pi_1$)
 7:         Run($\Pi_2$)
 8:         if $N(\Pi_1, \sigma_{out}) = N(\Pi_2, \sigma_{out}) \vee \overline{N}(\Pi_1, \sigma_{out}) = \overline{N}(\Pi_2, \sigma_{out})$ then
 9:             UpdateWeights($\Pi_1$)
10:             UpdateWeights($\Pi_2$)
11:         end if
12:     end while
13: end function
```

We study the efficiency of the function *Synchronize* with respect to the parameters K and N of the two ASNTPM P Systems. We note that both systems have the same parameters K, N and L. We denote by $T\,(Synchronize)$ the number of steps taken by the function *Synchronize*.

Hypothesis 1. The number of steps taken by the function *Synchronize* to synchronize two ASNTPM P Systems is quadratic in the parameter K of the inputs:

$$T\,(Synchronize) = 0.8K^2 - 30K + 1184 \tag{2}$$

Experiment. We run the algorithm for 50 times and compute the mean of the results for each $K \in \{4, 8, 16, 32, 64, 128, 256\}$ with $L = 256$ and $N = 128$. The results are presented in Table 1 and Figure 2.

Table 1. The efficiency of Algorithm 5 with respect to K

K	4	8	16	32	64	128	256
T(Synchronize)	158.08	286.66	623.34	1443.48	3532.3	10253.7	47492.06

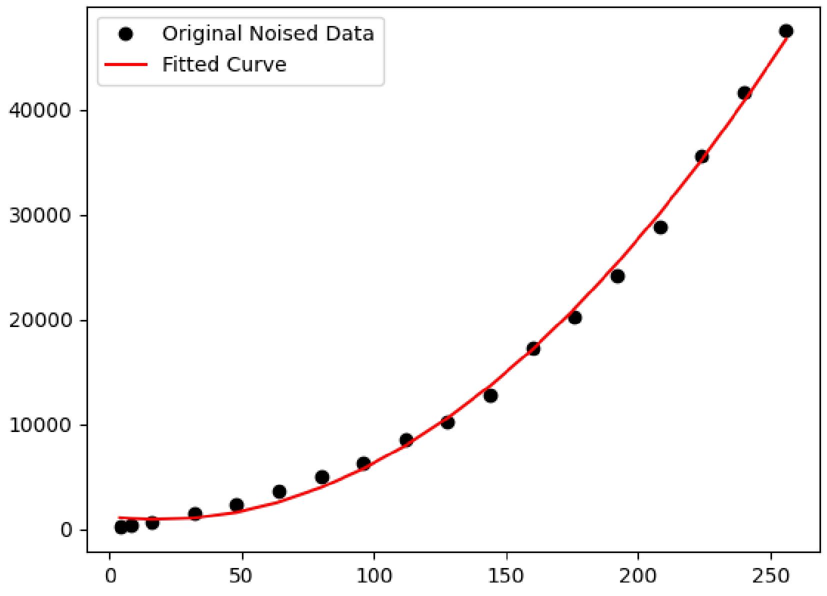

Fig. 2. The number of iterations of Algorithm 5 with respect to K

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best polynomial that fits the data and obtained (2).

Hypothesis 2. The number of steps taken by the function *Synchronize* to synchronize two ASNTPM P Systems is linear in the parameter N of the inputs:

$$T\left(Synchronize\right) = 27N + 191 \tag{3}$$

Experiment. We run the algorithm for 50 times and compute the mean of the results for each $N \in \{4, 8, 16, 32, 64, 128, 256\}$ with $L = 256$ and $K = 64$. The results are presented in Table 2 and Figure 3.

Table 2. The efficiency of Algorithm 5 with respect to N

N	4	8	16	32	64	128	256
T(Synchronize)	277.42	483.64	744.28	1173.58	1960.0	3594.54	7441.92

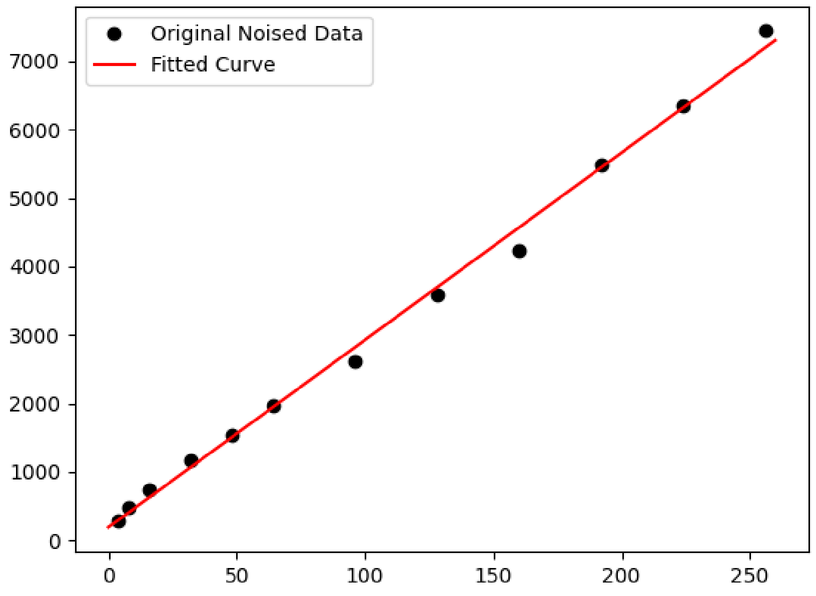

Fig. 3. The number of iterations of Algorithm 5 with respect to N

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best polynomial that fits the data and obtained (3).

5 Attacks on the synchronization process

In this section, we will examine three algorithms through a series of experiments that aim to synchronize two ASNTPMs without mutual learning. These algorithms are inspired by various attacks on key agreement protocols based on TPMs [22, 44, 46]. All algorithms received as inputs three ASNTPMs: Π_1, Π_2 and Π_3. The purpose of each algorithm is to synchronize Π_3 with Π_1 without mutual learning in the time frame in which Π_1 and Π_2 synchronize with mutual learning using Algorithm 5.

Algorithm 6 presents the naive solution inspired by [22] while Algorithm 7 presents the geometric solution inspired by [46]. The genetic attack presented in [44] is exponential in K so we do not include it here. We denote by $\rho\left(\Pi_x, \Pi_y\right)$ the synchronization percentage between Π_x and Π_y after the execution of the synchronization algorithm.

5.1 The naive attack

The function *NaiveAttack* stops execution when Π_1 and Π_2 are fully synchronized. The algorithm returns the synchronization percentage between Π_1 and Π_3. Hypotheses 3 and 4 capture the relation between the parameters K and N of the ASNTPM P systems and $\rho(\Pi_1, \Pi_3)$.

Algorithm 6 The naive attack on the synchronization process

```
1: function NaiveAttack($\Pi_1, \Pi_2, \Pi_3$)
2:     while SynchronizationPercentage($\Pi_1, \Pi_2$) $\neq 1$ do
3:         $x \overset{R}{\leftarrow} \mathbb{Z}^{KN}$
4:         Initialize($\Pi_1, x$)
5:         Initialize($\Pi_2, x$)
6:         Initialize($\Pi_3, x$)
7:         Run($\Pi_1$)
8:         Run($\Pi_2$)
9:         Run($\Pi_3$)
10:        if $N(\Pi_1, \sigma_{out}) = N(\Pi_2, \sigma_{out}) \vee \overline{N}(\Pi_1, \sigma_{out}) = \overline{N}(\Pi_2, \sigma_{out})$ then
11:            UpdateWeights($\Pi_1$)
12:            UpdateWeights($\Pi_2$)
13:            if $N(\Pi_1, \sigma_{out}) = N(\Pi_3, \sigma_{out}) \vee \overline{N}(\Pi_1, \sigma_{out}) = \overline{N}(\Pi_3, \sigma_{out})$ then
14:                UpdateWeights($\Pi_3$)
15:            end if
16:        end if
17:    end while
18:    return SynchronizationPercentage($\Pi_1, \Pi_3$)
19: end function
```

Hypothesis 3. The synchronization percentage between Π_1 and Π_3 after running the function *NaiveAttack* drops exponentially in K according to (4).

$$\rho(\Pi_1, \Pi_3) = 1.27e^{-0.11K} + 0.04 \tag{4}$$

Experiment. We run the algorithm for 50 times and compute the mean of the results for each $K \in \{4, 8, 16, 32, 64, 128, 256\}$ with $L = 256$ and $N = 128$. The results are presented in Table 3 and Figure 4.

Table 3. $\rho(\Pi_1, \Pi_3)$ using Algorithm 6 with respect to K

K	4	8	16	96	128	192	256
$\rho(\Pi_1, \Pi_3)$	0.83	0.53	0.23	0.06	0.05	0.04	0.02

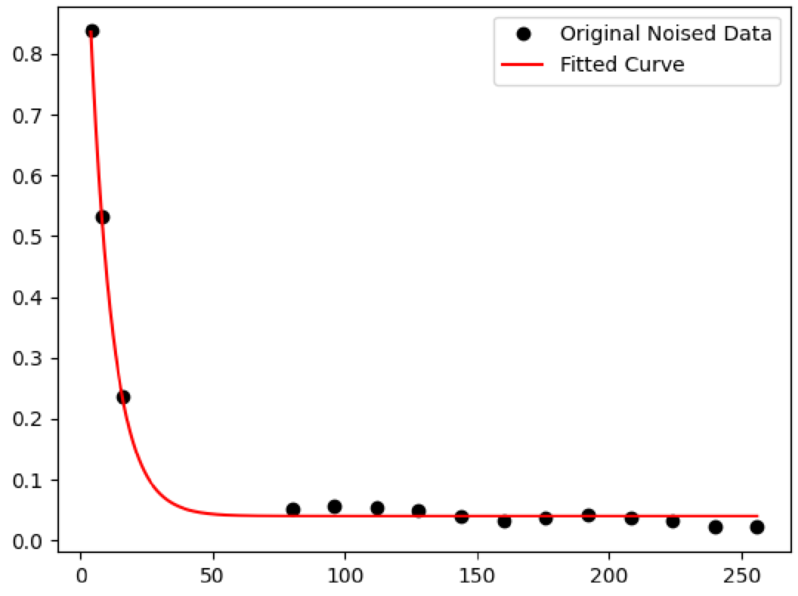

Fig. 4. $\rho\left(\Pi_1, \Pi_3\right)$ using Algorithm 6 with respect to K

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best power function that fits the data and obtained (4).

Hypothesis 4. The synchronization percentage between Π_1 and Π_3 after running the function $NaiveAttack$ drops in N according to (5).

$$\rho\left(\Pi_1, \Pi_3\right) = 0.02e^{-0.03N} + 0.06 \tag{5}$$

Experiment. We run the algorithm for 50 times and compute the mean of the results for each $N \in \{4, 8, 16, 32, 64, 128, 256\}$ with $L = 256$ and $K = 64$. The results are presented in Table 4 and Figure 5.

Table 4. $\rho\left(\Pi_1, \Pi_3\right)$ using Algorithm 6 with respect to N

K	4	8	16	96	128	192	256
$\rho\left(\Pi_1, \Pi_3\right)$	0.28	0.25	0.18	0.09	0.08	0.07	0.06

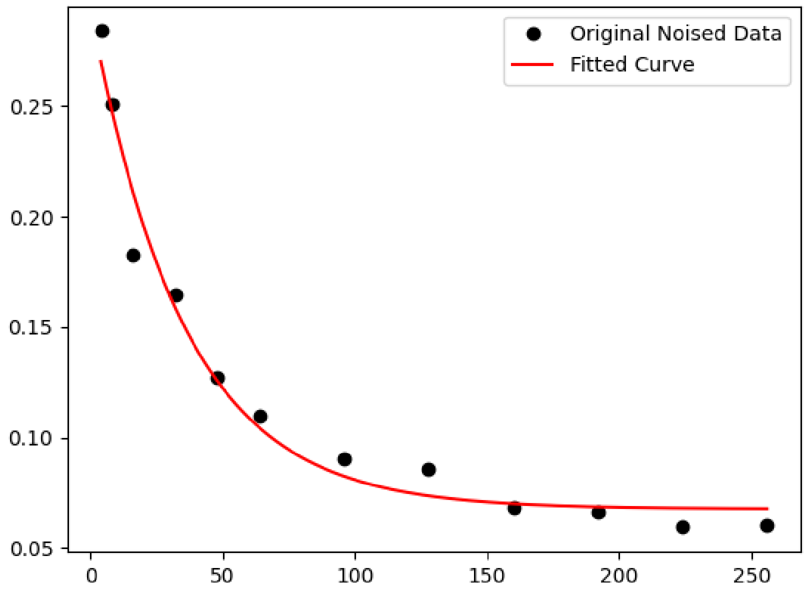

Fig. 5. $\rho(\Pi_1, \Pi_3)$ using Algorithm 6 with respect to N

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best power that fits the data and obtained (5).

5.2 The geometric attack

Another solution for the synchronization of three ASNTPM P Systems is presented in Algorithm 7. This solution is inspired by the geometric attack on neural cryptography presented in [22].

The idea of this solution is to interpret the weights and the input associated with each hidden neuron as points in a $N-dimensional$ discrete space. When the outputs of Π_1 and Π_2 are the same but the output of Π_3 is different then at least one hidden neuron of Π_3 has the wrong number of spikes or anti-spikes.

To correct this error, we compute the distance between the input and the weights associated with each hidden neuron of Π_3. The hidden neuron which presents the minimum distance will have the number of spikes and anti-spikes inverted. The output of Π_3 will be set to the output of Π_1 and the weights of Π_3 will be updated using the new configuration.

Although this solution is more efficient than the one presented in Algorithm 6, the synchronization percentage still decreases as K or N increases.

Algorithm 7 The geometric solution for the synchronization of three ASNTPM P Systems

```
function GeometricAttack($\Pi_1, \Pi_2, \Pi_3$)
    while SynchronizationPercentage($\Pi_1, \Pi_2$) $\neq 1$ do
        $x \overset{R}{\leftarrow} \mathbb{Z}^{KN}$
        Initialize($\Pi_1, x$)
        Initialize($\Pi_2, x$)
        Initialize($\Pi_3, x$)
        Run($\Pi_1$)
        Run($\Pi_2$)
        Run($\Pi_3$)
        if $N(\Pi_1, \sigma_{out}) = N(\Pi_2, \sigma_{out}) \vee \overline{N}(\Pi_1, \sigma_{out}) = \overline{N}(\Pi_2, \sigma_{out})$ then
            UpdateWeights($\Pi_1$)
            UpdateWeights($\Pi_2$)
            if $N(\Pi_1, \sigma_{out}) = N(\Pi_3, \sigma_{out}) \vee \overline{N}(\Pi_1, \sigma_{out}) = \overline{N}(\Pi_3, \sigma_{out})$ then
                UpdateWeights($\Pi_3$)
            end if
            if $N(\Pi_1, \sigma_{out}) \neq N(\Pi_3, \sigma_{out}) \wedge \overline{N}(\Pi_1, \sigma_{out}) \neq \overline{N}(\Pi_3, \sigma_{out})$ then
                $distance = \|W_{\Pi_3}[1] - x\|$
                $minimum = distance$
                $index = 1$
                for $i = 2;\ i \leq K;\ i = i + 1$ do
                    $distance = \|W_{\Pi_3}[i] - x\|$
                    if $distance < minimum$ then
                        $mininum = distance$
                        $index = i$
                    end if
                end for
                $aux = \overline{N}(\Pi_3, \sigma_{h_{index}})$
                $\overline{N}(\Pi_3, \sigma_{h_{index}}) = N(\Pi_3, \sigma_{h_{index}})$
                $N(\Pi_3, \sigma_{h_{index}}) = aux$
                $\overline{N}(\Pi_3, \sigma_{output}) = \overline{N}(\Pi_1, \sigma_{output})$
                $N(\Pi_3, \sigma_{output}) = N(\Pi_1, \sigma_{output})$
                UpdateWeights($\Pi_3$)
            end if
        end if
    end while
    return SynchronizationPercentage($\Pi_1, \Pi_3$)
end function
```

Hypothesis 5. The synchronization percentage between Π_1 and Π_3 after running the function *GeometricAttack* drops in K according to (6).

$$\rho\left(\Pi_1, \Pi_3\right) = 0.91e^{-0.004K} \tag{6}$$

Experiment. We run the algorithm for 50 times and compute the mean of the results for each $K \in \{4, 8, 16, 32, 64, 128, 256, 512\}$ with $L = 256$ and $N = 128$. The results are presented in Table 5 and Figure 6.

Table 5. $\rho\left(\Pi_1, \Pi_3\right)$ using Algorithm 7 with respect to K

K	4	8	16	32	64	128	256	512
$\rho\left(\Pi_1, \Pi_3\right)$	0.95	0.88	0.82	0.73	0.65	0.55	0.27	0.05

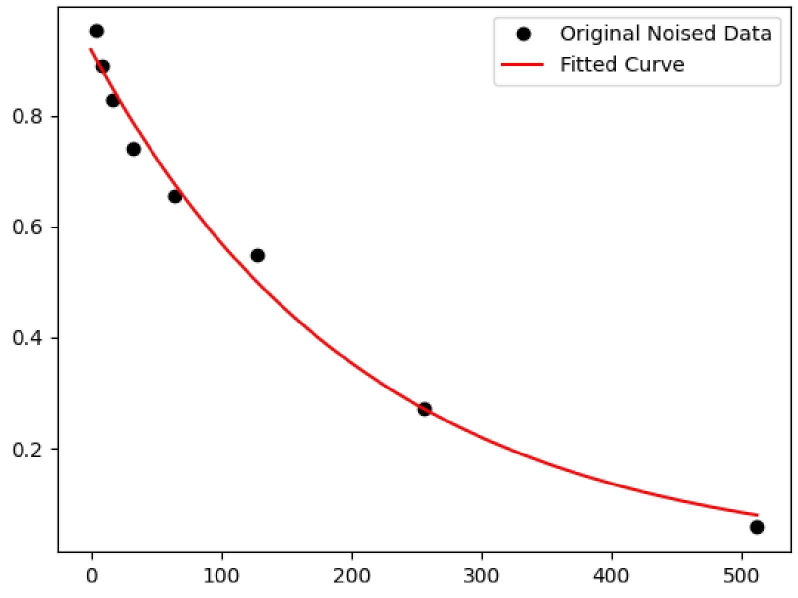

Fig. 6. $\rho\left(\Pi_1, \Pi_3\right)$ using Algorithm 7 with respect to K

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best exponential function that fits the data and obtained (6).

Hypothesis 6. The synchronization percentage between Π_1 and Π_3 after running the function *GeometricAttack* drops in N according to (7).

$$\rho\left(\Pi_1, \Pi_3\right) = 0.99e^{-0.003N} \tag{7}$$

Experiment. We run the algorithm for 50 times and compute the mean of the results for each $N \in \{4, 8, 16, 32, 64, 128, 256, 512\}$ with $L = 256$ and $K = 64$. The results are presented in Table 6 and Figure 7.

Table 6. $\rho\left(\Pi_1, \Pi_3\right)$ using Algorithm 7 with respect to N

N	4	8	16	32	64	128	256	512
$\rho\left(\Pi_1, \Pi_3\right)$	0.98	0.97	0.95	0.90	0.78	0.65	0.46	0.23

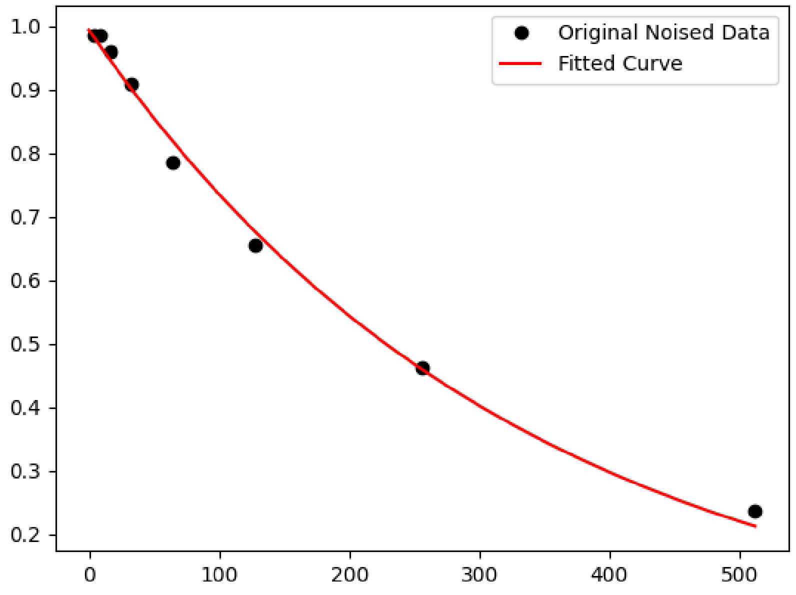

Fig. 7. $\rho\left(\Pi_1, \Pi_3\right)$ using Algorithm 7 with respect to N

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best exponential function that fits the data and obtained (6).

5.3 The majority attack

Let the ordered set $H_{\Pi} = \left(N(\Pi, \sigma_{h_1}), \overline{N}(\Pi, \sigma_{h_1}), N(\Pi, \sigma_{h_2}), \overline{N}(\Pi, \sigma_{h_2}), ..., N(\Pi, \sigma_{h_K}), \overline{N}(\Pi, \sigma_{h_K})\right)$ be the hidden state of the ASNTPM P System Π. Let $S = \{H_{\Pi_1}, H_{\Pi_2}, ..., H_{\Pi_M}\}$ be the set of hidden states of M ASNTPM P Systems. We denote by $F_S\left(H_{\Pi_i}\right)$ the frequency of the element H_{Π_i} in the set S. Given two ASNTPM P Systems Π_A and Π_B we can synchronize them in parallel with each of the M ASNTPM Systems Π_i, for each $1 \leq i \leq M$ using the geometric solution. Similar to [46] we could try to design a solution that uses the frequency $F_S\left(H_{\Pi_i}\right)$ to synchronize Π_A and Π_B with at least one of the M ASNTPM P Systems. However, this is not possible given the fact that for ASNTPM P Systems there exist certain values of K, s.a. $F_S\left(H_{\Pi_i}\right) = \frac{1}{M}$, $\forall 1 \leq i \leq M$. This is illustrated by hypotheses 7 and 8.

Hypothesis 7. Given two ASNTPM P Systems Π_A and Π_B and a set of M ASNTPM P Systems $\{\Pi_1, \Pi_2, ..., \Pi_M\}$, the mean of the frequencies $F_S\left(H_{\Pi_i}\right)$ after each iteration of *GeometricAttack* (Π_A, Π_B, Π_i), $1 \leq i \leq M$ converges to $\frac{1}{M}$ as K increases according to (8).

$$\frac{1}{M}\sum_{i=1}^{M} F_S\left(H_{\Pi_i}\right) = 1.13e^{-0.06K} \tag{8}$$

Experiment. We run *GeometricAttack* (Π_A, Π_B, Π_i) in parallel for each $1 \leq i \leq M$ and averaged the result. The procedure was repeated 50 times and the mean of the results were computed for each $K \in \{4, 8, 16, 32, 64, 128, 256, 512\}$ with $L = 256$, $N = 128$ and $M = 128$. The results are presented in Table 7 and Figure 8. Let $\mu_{F_S} = \frac{1}{M}\sum_{i=1}^{M} F_S\left(H_{\Pi_i}\right)$.

Table 7. The average μ_{F_S} with respect to K

K	4	8	16	32	64	128	256	512
μ_{F_S}	0.85	0.66	0.36	0.13	0.018	0.00786	0.0078125	0.0078125

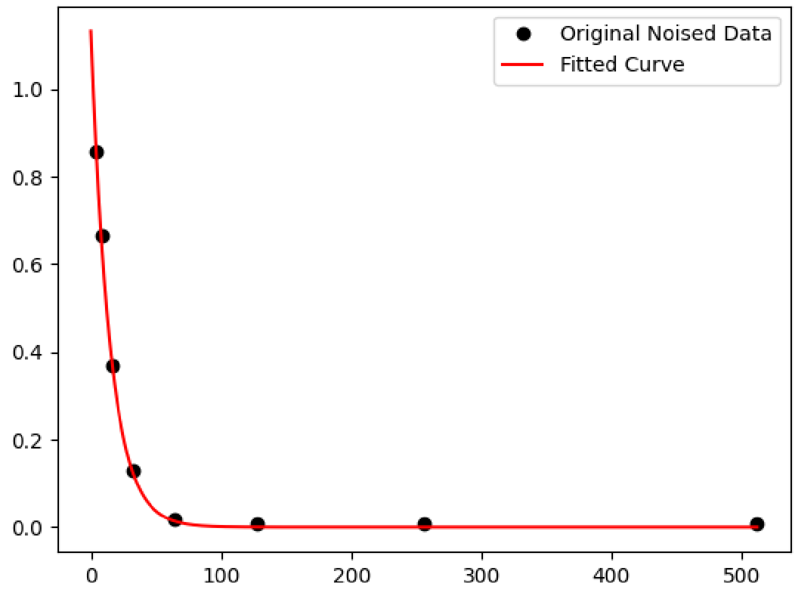

Fig. 8. μ_{F_S} with respect to K

Using the least square method with the Levenberg-Marquardt algorithm on the values recorded during the experiment we computed the best exponential function that fits the data and obtained (8).

6 Conclusions and further directions of research

In this paper, we analyzed the security of the neural key agreement protocol proposed in [40]. The protocol proposed by the authors is based on a new type of TPM called the Anti-Spiking Neural Tree Parity Machine. Unlike classical TPM, the model of the neuron used by ASNTPM is inspired by Spiking Neural P systems with anti-spikes. We adopt four different types of attacks on TPM-based protocols: the naive attack, the geometric attack, the majority attack and the genetic attack. We showed through a series of experiments that increasing the number of neurons decreases exponentially the percentage of the key recovered by the attacker. This growth in the number of neurons implies only a polynomial increase in the running time. From a cryptographic perspective, this behavior is similar to trapdoor problems on which the currently used cryptosystems are based.

A further direction of research is to formalize the hard problem on which the security of the TPM-based key agreement protocol is based. Analyzing the security of a specific protocol is dependent on the security model. Constructing a hard problem enables the creation of many cryptographic primitives whose security can be proved by reduction to the underlining hard problem.

Another direction of research implies analyzing the security of the protocol using another security model in which the attacker can alter the messages exchanged by the legitimate parties. The current protocol is insecure in such a security model given the fact that any third party can mount a Man-in-the-Middle (MitM) attack.

The third direction of research is to construct neural key agreement protocols for groups. This is particularly important because most applications for secure communications are designed for group messaging.

References

1. Abu-Zidan, Y., Nguyen, K., Mendis, P., Setunge, S., Adeli, H.: Design of a smart prefabricated sanitising chamber for covid-19 using computational fluid dynamics. Journal of Civil Engineering and Management **27**(2), 139–148 (2021)
2. Adeli, H., Kim, H.: Wavelet-based vibration control of smart buildings and bridges. CRC Press (2022)
3. Adorna, H.N.: Computing with SN P systems with I/O mode. Journal of Membrane Computing **2**(4), 230–245 (2020)
4. Alagic, G., Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status report on the first round of the nist post-quantum cryptography standardization process (2019)
5. Alhazov, A., Freund, R., Ivanov, S.: P systems with limited number of objects. Journal of Membrane Computing **3**(1), 1–9 (2021)
6. Alhazov, A., Sburlan, D.: Static sorting P systems. In: Applications of Membrane Computing, pp. 215–252. Springer (2006)
7. Allam, A.M., Abbas, H.M.: Improved security of neural cryptography using don't-trust-my-partner and error prediction. In: 2009 International Joint Conference on Neural Networks. pp. 121–127. IEEE (2009)
8. Allam, A.M., Abbas, H.M.: On the improvement of neural cryptography using erroneous transmitted information with error prediction. IEEE transactions on neural networks **21**(12), 1915–1924 (2010)
9. Cavaliere, M., Ibarra, O.H., Păun, G., Egecioglu, O., Ionescu, M., Woodworth, S.: Asynchronous spiking neural P systems. Theoretical Computer Science **410**(24-25), 2352–2364 (2009)
10. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. Journal of Cryptology **33**, 1914–1983 (2020)
11. Dierks, T., Allen, C.: The TLS protocol version 1.0. Tech. rep. (1999)
12. Dong, T., Huang, T.: Neural cryptography based on complex-valued neural network. IEEE Transactions on Neural Networks and Learning Systems **31**(11), 4999–5004 (2019)

13. Ganbaatar, G., Nyamdorj, D., Cichon, G., Ishdorj, T.O.: Implementation of RSA cryptographic algorithm using SN P systems based on HP/LP neurons. Journal of Membrane Computing **3**(1), 22–34 (2021)
14. Gheorghe, M., Lefticaru, R., Konur, S., Niculescu, I.M., Adorna, H.N.: Spiking neural P systems: matrix representation and formal verification. Journal of Membrane Computing **3**(2), 133–148 (2021)
15. Goldreich, O.: Foundations of cryptography: volume 2, basic applications. Cambridge university press (2009)
16. Hassanpour, A., Moradikia, M., Adeli, H., Khayami, S.R., Shamsinejadbabaki, P.: A novel end-to-end deep learning scheme for classifying multi-class motor imagery electroencephalography signals. Expert Systems **36**(6), e12494 (2019)
17. Hinze, T., Happe, H., Henderson, A., Nicolescu, R.: Membrane computing with water. Journal of Membrane Computing **2**(2), 121–136 (2020)
18. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae **71**(2-3), 279–308 (2006)
19. Ipate, F., Lefticaru, R., Mierlă, L., Cabrera, L.V., Han, H., Zhang, G., Dragomir, C., Jiménez, M.J.P., Gheorghe, M.: Kernel P systems: Applications and implementations. In: Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2013. pp. 1081–1089. Springer (2013)
20. Jeong, S., Park, C., Hong, D., Seo, C., Jho, N.: Neural cryptography based on generalized tree parity machine for real-life systems. Security and Communication Networks **2021** (2021)
21. Kanter, I., Kinzel, W., Kanter, E.: Secure exchange of information by synchronization of neural networks. EPL (Europhysics Letters) **57**(1), 141 (2002)
22. Klimov, A., Mityagin, A., Shamir, A.: Analysis of neural cryptography. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 288–298. Springer (2002)
23. Kumar, A., Garhwal, S.: State-of-the-art survey of quantum cryptography. Archives of Computational Methods in Engineering **28**, 3831–3868 (2021)
24. Lv, Z., Yang, Q., Peng, H., Song, X., Wang, J.: Computational power of sequential spiking neural P systems with multiple channels. Journal of Membrane Computing **3**(4), 270–283 (2021)
25. Mammone, N., Ieracitano, C., Adeli, H., Morabito, F.C.: Autoencoder filter bank common spatial patterns to decode motor imagery from eeg. IEEE Journal of Biomedical and Health Informatics (2023)
26. Mayne, R., Phillips, N., Adamatzky, A.: Towards experimental P-systems using multivesicular liposomes. Journal of Membrane Computing **1**(1), 20–28 (2019)
27. Mirzaei, G., Adeli, H.: Machine learning techniques for diagnosis of alzheimer disease, mild cognitive disorder, and other types of dementia. Biomedical Signal Processing and Control **72**, 103293 (2022)
28. Mislovaty, R., Perchenok, Y., Kanter, I., Kinzel, W.: Secure key-exchange protocol with an absence of injective functions. Physical Review E **66**(6), 066102 (2002)
29. Mosavi, A.H., Mohammadzadeh, A., Rathinasamy, S., Zhang, C., Reuter, U., Levente, K., Adeli, H.: Deep learning fuzzy immersion and invariance control for type-i diabetes. Computers in Biology and Medicine **149**, 105975 (2022)
30. Murugappan, M., Murugesan, L., Jerritta, S., Adeli, H.: Sudden cardiac arrest (sca) prediction using ecg morphological features. Arabian Journal for Science and Engineering **46**, 947–961 (2021)

31. Nogay, H.S., Adeli, H.: Machine learning (ml) for the diagnosis of autism spectrum disorder (asd) using brain imaging. Reviews in the Neurosciences **31**(8), 825–841 (2020)
32. Nogay, H.S., Adeli, H.: Diagnostic of autism spectrum disorder based on structural brain mri images using, grid search optimization, and convolutional neural networks. Biomedical Signal Processing and Control **79**, 104234 (2023)
33. Pan, L., Păun, G.: Spiking neural P systems with anti-spikes. International Journal of Computers Communications & Control **4**(3), 273–282 (2009)
34. Pan, L., Păun, G., Zhang, G., Neri, F.: Spiking neural P systems with communication on request. International Journal of Neural Systems **27**(08), 1750042 (2017)
35. Pan, L., Wang, J., Hoogeboom, H.J.: Spiking neural P systems with astrocytes. Neural Computation **24**(3), 805–825 (2012)
36. Pan, X., Yang, T., Xiao, Y., Yao, H., Adeli, H.: Vision-based real-time structural vibration measurement through deep-learning-based detection and tracking methods. Engineering Structures **281**, 115676 (2023)
37. Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000)
38. Peng, W., Wang, B., Hu, F., Wang, Y., Fang, X., Chen, X., Wang, C.: Factoring larger integers with fewer qubits via quantum annealing with optimized parameters. SCIENCE CHINA Physics, Mechanics & Astronomy **62**, 1–8 (2019)
39. Pereira, D.R., Piteri, M.A., Souza, A.N., Papa, J.P., Adeli, H.: Fema: A finite element machine for fast learning. Neural Computing and Applications **32**, 6393–6404 (2020)
40. Plesa, M.I., Gheoghe, M., Ipate, F., Zhang, G.: A key agreement protocol based on spiking neural p systems with anti-spikes. Journal of Membrane Computing pp. 1–11 (2022)
41. Rafiei, M.H., Gauthier, L.V., Adeli, H., Takabi, D.: Self-supervised learning for electroencephalography. IEEE Transactions on Neural Networks and Learning Systems (2022)
42. Rescorla, E.: The transport layer security (tls) protocol version 1.3. Tech. rep. (2018)
43. Ruttor, A., Kinzel, W., Kanter, I.: Neural cryptography with queries. Journal of Statistical Mechanics: Theory and Experiment **2005**(01), P01009 (2005)
44. Ruttor, A., Kinzel, W., Naeh, R., Kanter, I.: Genetic attack on neural cryptography. Physical Review E **73**(3), 036121 (2006)
45. Salguero Dorokhin, É., Fuertes, W., Lascano, E.: On the development of an optimal structure of tree parity machine for the establishment of a cryptographic key. Security and Communication Networks **2019** (2019)
46. Shacham, L.N., Klein, E., Mislovaty, R., Kanter, I., Kinzel, W.: Cooperating attackers in neural cryptography. Physical Review E **69**(6), 066137 (2004)
47. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review **41**(2), 303–332 (1999)
48. Song, T., Pan, L., Wu, T., Zheng, P., Wong, M.D., Rodríguez-Patón, A.: Spiking neural P systems with learning functions. IEEE Transactions on Nanobioscience **18**(2), 176–190 (2019)
49. Song, T., Rodríguez-Patón, A., Zheng, P., Zeng, X.: Spiking neural P systems with colored spikes. IEEE Transactions on Cognitive and Developmental Systems **10**(4), 1106–1115 (2017)
50. Song, X., Wang, J., Peng, H., Ning, G., Sun, Z., Wang, T., Yang, F.: Spiking neural P systems with multiple channels and anti-spikes. Biosystems **169**, 13–19 (2018)

51. Stypiński, M., Niemiec, M.: Synchronization of Tree Parity Machines using nonbinary input vectors. arXiv preprint arXiv:2104.11105 (2021)
52. Stypiński, M., Niemiec, M.: Impact of nonbinary input vectors on security of tree parity machine. In: Multimedia Communications, Services and Security: 11th International Conference, MCSS 2022, Kraków, Poland, November 3–4, 2022, Proceedings. pp. 94–103. Springer (2022)
53. Vasile, R., Gheorghe, M., Niculescu, I.M.: Breaking RSA Encryption Protocol with Kernel P Systems (2023)
54. Wang, H., Zhou, K., Zhang, G., Paul, P., Duan, Y., Qi, H., Rong, H.: Application of Weighted Spiking Neural P Systems with Rules on Synapses for Breaking RSA Encryption. International Journal of Unconventional Computing **15**(1-2), 37–58 (2020)
55. Wang, X., Song, T., Zheng, P., Hao, S., Ma, T.: Spiking neural P systems with antispikes and without annihilating priority. Romanian Journal of Science and Technology **20**(1), 32–41 (2017)
56. Wu, T., Jiang, S.: Spiking neural P systems with a flat maximally parallel use of rules. Journal of Membrane Computing **3**(3), 221–231 (2021)
57. Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking neural P systems with polarizations. IEEE Transactions on Neural Networks and Learning Systems **29**(8), 3349–3360 (2017)
58. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Unconventional Models of Computation, UMC'2K, pp. 289–301. Springer (2001)
59. Zhang, G., Zhang, X., Rong, H., Paul, P., Zhu, M., Neri, F., Ong, Y.S.: A layered spiking neural system for classification problems. International Journal of Neural Systems p. 2250023 (2022)

Stochastic virus machines

Antonio Ramírez-de-Arellano[1,2], José Antonio Rodríguez Gallego[3],
David Orellana-Martín[1,2], Sergiu Ivanov[4]

[1]Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {aramirezdearellano,dorellana}@us.es
[2]SCORE Laboratory, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[3]FQM120: Modelado Matemático y Simulación de
Sistemas Medioambientales,
Universidad de Sevilla, Spain
E-mail: jrodriguez14@us.es
[4]IBISC Laboratory, Université Paris-Saclay, Univ Évry
91020, Évry-Courcouronnes, France
E-mail : sergiu.ivanov@univ-evry.fr

Summary. Virus machines are an abstract computing device capturing viral reproduction and transmission as a computation. Virus machines feature hosts containing viruses, whose evolution is governed by the associated instruction graph, and which migrate via the channels connecting the hosts. In this paper, we propose using virus machines as a modelling device to represent real-world virus propagation. We start with the widely used SIR model, numerically accounting for susceptible (S), infected (I), and recovered (R) individuals, and propose two virus machines representing two simple situations with respectively two and three possible locations for the agents.

Key words: Virus machines, pandemics modelling, SIR model.

1 Introduction

This work can be considered as a contribution to the area of *Natural Computing*, which is a field of research that investigates both human-designed computing inspired by nature and computing that occurs in nature.

In virology, a virus is an infectious agent of small size and simple composition that can only reproduce after infecting a host cell. All animal, plant and protist species on the planet can be and have been infected by viruses. Indeed, biologists estimate that we have about 350 trillion viruses living in our bodies [1], which is

10 times the number of bacteria and cells in the human body. This leads us to believe that it would be very interesting to study this biological structure from a computational point of view. For more details on viruses, see [3].

This study focuses on a new computing paradigm, introduced in [2], based on the transmissions and replications of viruses. This innovative paradigm provides non-deterministic computing devices that consist of several biological *hosts* connected with each another by *directed channels*. The viruses are contained in these hosts and will be able to both transmit and replicate themselves passing through these channels. This processes are controlled by several instructions, which are attached to the channels. These systems can be considered as a heterogeneous network that consists of:

- A *virus transmission network*: a weighted directed graph, wherein each node represents a *host* and each arc represents a *transmission channel* through which viruses can transmit between hosts or exit to the environment. In addition, each arc has an associated weight (a natural number $w > 0$) , which indicates the number of viruses that will be replicated in each transmission.
- An *instruction transfer network*: a weighted directed graph, wherein each node represents a *control instruction unit* and each edge represents an optional *instruction transfer path* with a positive integral weight.
- An *instruction-channel control network*: an undirected graph, wherein each node represents either a control instruction or a transmission channel and each edge represents a relationship between an instruction and a channel.

The computing models of this paradigm are universal (equivalent in power to Turing machines), demonstrated by generating Diophantine sets [4], by computing partial recursive functions [5] and by simulating register machines [2].

2 Stochastic Virus Machines

In order to fit an extension of the Virus Machines (VM for short) with exciting instructions for modelling SIR model, we propose that "excitation" of the hosts can be stochastic, *i.e.*, the weight of the channels between hosts can be a tuple (a, p) where a is the capacity of replication if the channel is opened (as basic VM paradigm) and p is the probability of opening that channel in case the origin host is excited[1].

3 Modelling

The basic idea of this extension is that, from now on, viruses will represent the population and hosts will represent not only a place but also a status, so the num-

[1] Notice that only one of the channels could be opened, in order to avoid further complexity with the number of viruses and the meaning of the probability.

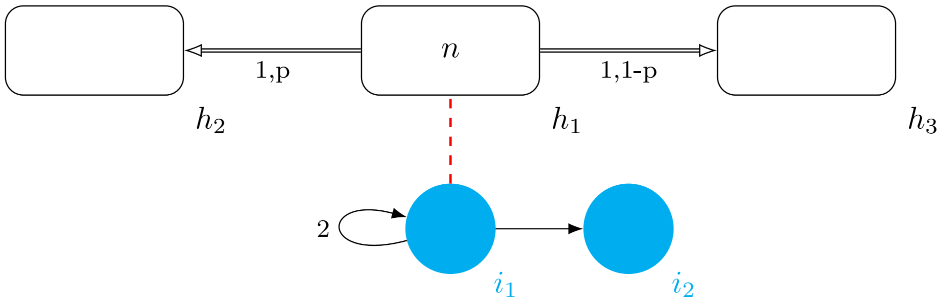

Fig. 1. Initial configuration of a SVM.

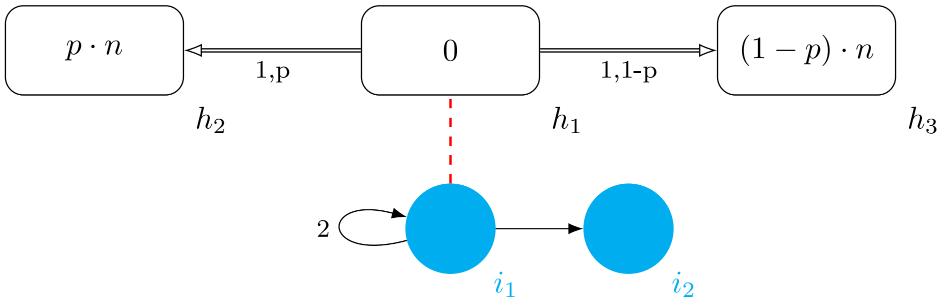

Fig. 2. Average halting configuration of a SVM.

ber of viruses a host is containing at some instant t, $h_{Place_{Status}}$, is the population located at *Place* in compartment *Status* at instant t.

Specifically, for the classical SIR model (susceptible-infected-removed) we will have two kind of hosts:

i) the hosts which contain the people who are susceptible of being infected (**S**),
ii) the infected population (**I**).

In addition, by considering that the recovered people (**R**) in the SIR model is "passive" (if and only if recovered people cannot be re-infected), we can take advantage of the passive behavior of the environment, so that the number of viruses sent to environment represents the number of recovered people.

4 A case of study: Pandemics

A *pandemic* is an epidemic that occurs over a wide geographical area, affecting a significant proportion of the population. It is characterized by the rapid spread of an infectious disease, often caused by a novel pathogen, which has the potential to cause severe illness or death.

In this section, we present a SIR computational model based on Stochastic Virus Machines. SIR in an acronym: **S** stands for the *susceptible population*, those who are not yet infected, but may become infected; **I** stands for the infected population, those who are ill and can transmit the disease, and **R** stands for the dead or recovered individuals that are removed from the infected population and cannot transmit the disease.

The SIR mathematical model for pandemics is an ODEs based model that has been used to understand the temporal transmission dynamics of the infection:

$$\frac{\partial S}{\partial t} = -p \cdot S \cdot I, \ \ \frac{\partial I}{\partial t} = p \cdot S \cdot I - r \cdot I, \ \frac{\partial R}{\partial t} = r \cdot I.$$

where S, I and R represent the number of individuals in the susceptible, infected and recovered compartments respectively, p is the transmission rate, which determines the rate at which susceptible individuals become infected, and r is the recovery rate, which determines the rate at which infected individuals recover and become immune to the disease. The parameter $(\frac{p}{r})$ is known as the basic reproduction number (R_0), which represents the average number of secondary infections produced by a single infected individual in a susceptible population. The SIR model assumes that the population is well-mixed and that the disease spreads through direct contact between individuals. It also assumes that individuals do not acquire natural immunity, and that there is no vaccination or treatment available for the disease.

Our case of study will be restricted to two physically separated places (e.g. home and supermarket). A susceptible person can be infected either in their way to the supermarket, or on their way back. The figure displays a global view of the case stated above.

4.1 Design of a SVM Modelling Pandemics

In this section, the model for the exposed case of study by using SVM is presented.

A first step in the approach of a bigger example, is to start with a simple one, let us suppose a SIR model of a population $N = S_0 + I_0$, where S_0 and I_0 are the initial population of susceptible and infected people resp. and two locations are defined: i) Home (H) and ii) Supermarket (J(amón)). Let us also suppose that the probability of being infected is p with $0 < p \leq 1$ and the probability of being recovered for an infected individual is $0 < r \leq 1$. We propose the SVM shown in Figure 3.

The idea behind this machine is as follows. The population is constantly moving back and forth from home to the supermarket. If there is an infected individual in a place A, then there would be a probability p that a susceptible person will be infected while going to the other place; and similarly, there would be a probability $1 - p$ that they will not be infected. If there are no infected people in that place, then the process is repeated for the other place, and if the other place also ran out of infected people, it leads to a halting configuration. Meanwhile, the infected

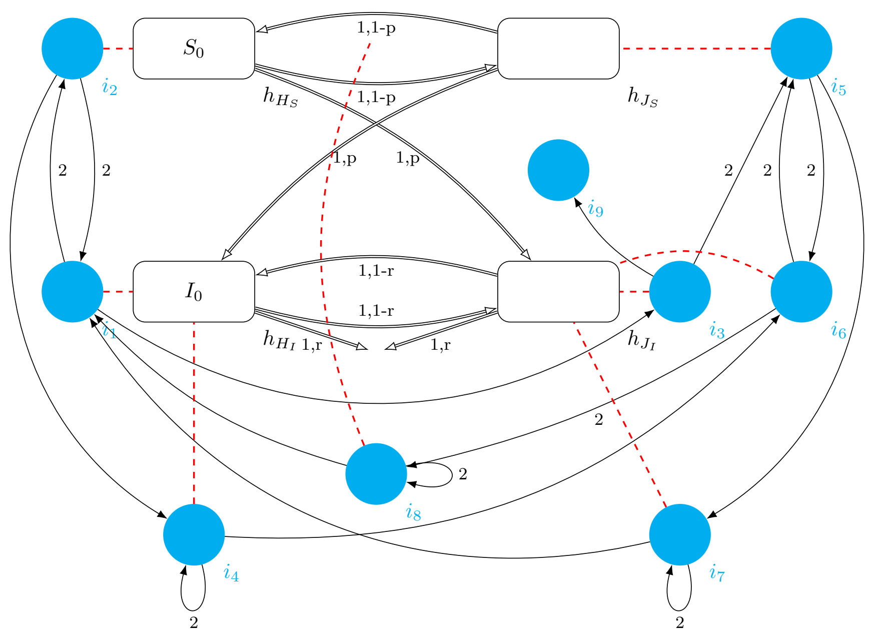

Fig. 3. A SVM modelling the SIR model with two locations

population can be recovered (sent to the environment) with probability r or can go to the other place infected with probability $(1 - p)$.

From now on, let us study a more complex example. Suppose the existence of an extra location D(isco) where the probability of being infected is even higher than in the supermarket. The global idea for this model is presented in Figure 4, and we will use the following notation:

- $0 \leq p_H, p_J, p_D \leq 1$: the probability of travelling form home to each location such that $p_H + p_J + p_D = 1$,
- $0 \leq p_{IH}, p_{IJ}, p_{ID} \leq 1$: the probability of being infected at each location,
- $0 \leq r_H, r_J, r_D \leq 1$: the probability of being recovered at each location.

The SVM modelling the SIR model is presented in two parts, the host graph in Figure 5, and the instruction graph in Figure 6 with a legend of the hosts/channels attached to each instruction.

4.2 Model Analysis

Now we will execute a thorough analysis of the above presented model, in order to provide as much insight as possible, starting with the expected behavior of the model and following with the limitations of the model, as it presents a certain bias, a fact which will be exposed shortly.

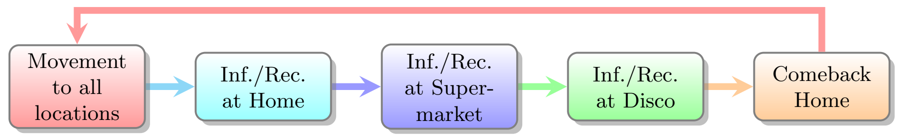

Fig. 4. Modules corresponding to the SIR model

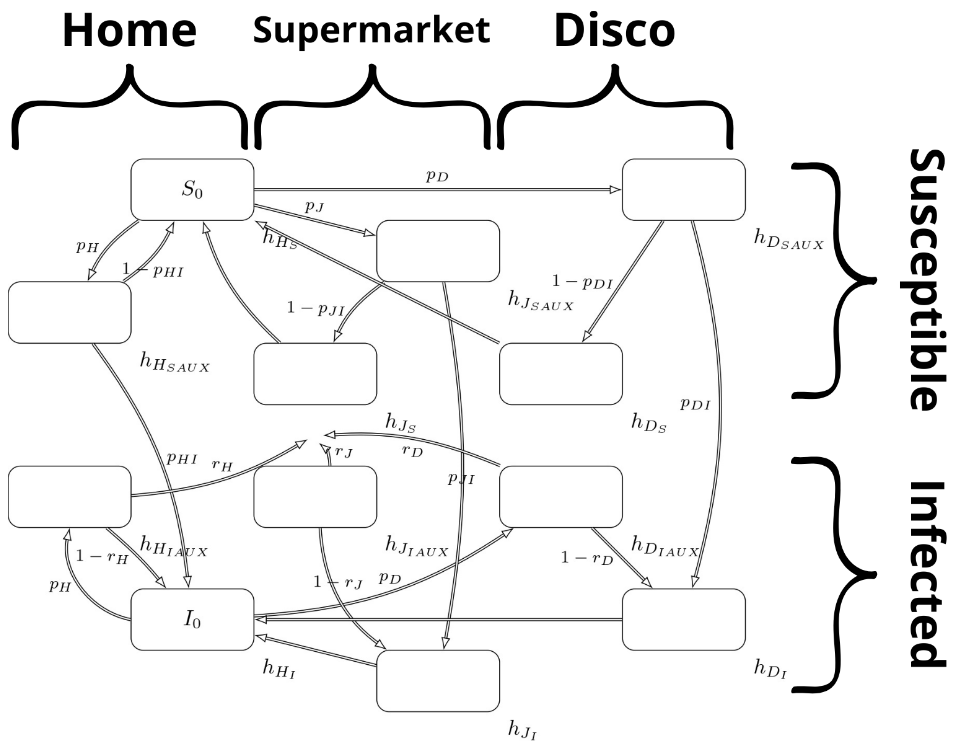

Fig. 5. A SVM modelling the SIR model (Hosts)

To begin with, as this is a really small model, some of its properties can be directly obtained, as exposed above. However, one which could be to a certain degree intuitive but not completely clear is the bias it has for the complete removal of the infected people in a large enough amount of time. Recall that the SIR model, in its most basic configuration, tends to reach a balance situation when only infected and recovered (which can not be infected again) interact, equivalent to a halting configuration in which infected become recovered and the virus finally disappears. This behavior is replicated by our model, as the halting configuration is, as we have already mentioned, one in which there are no infected people.

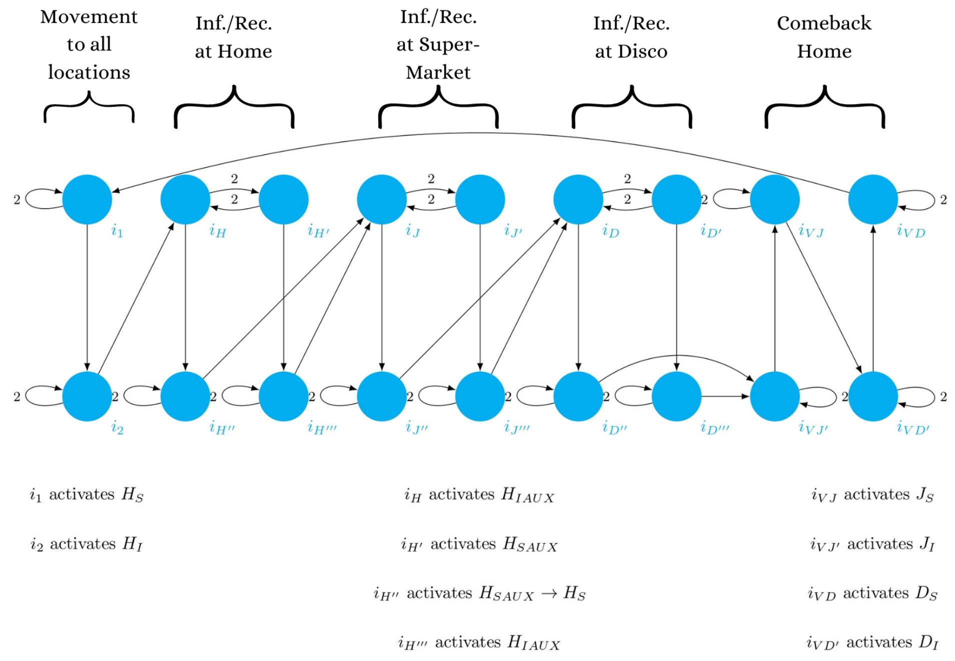

Fig. 6. A SVM modelling the SIR model (Instructions)

Lastly, a fact we must point out for its relevance is that, even if it is the first known modelling of the SIR model in a SVM, expected to be used as a minimal, educational example, and not suitable for real data analysis, an obvious problem of the presented model is its complexity. Generally, VM are not known for their simplicity, but when working with the stochastic variant, specially regarding the graphical representation used in Figure 3, we can assert that another representation is needed, in particular for more complex models.

5 Conclusion

With the aid of the Virus Machines (VM) several phenomena can be modelled, but the reach of these models is limited, especially for the modelling of more complex situations. With that in mind, in this article the Stochastic Virus Machines (SVM) are presented as a non-deterministic extension for the VM, providing more tools for modelling certain events, such as pandemics. To explore this fact, an example of SVM following the classical SIR model is shown and analyzed, pointing out both its suitability as an alternative for the traditional differential model and its complexity, an issue which is yet to be addressed. More precisely, two simple cases have been studied with interesting results, the second one can be easily extended for more locations or kind of populations (young, elder people, etc.).

Acknowledgements

The authors would like to thank the Organizing Committee of the 19th Brainstorming Week on Membrane Computing[2] (BWMC 2023) for organizing this fruitful event.

References

1. J.L. Mokili, F. Rohwer, B.E. Dutilh. Metagenomics and future perspectives in virus discovery. *Current Opinion in Virology*, **2**, 1 (2012), 63–77.
2. X. Chen, M.J. Pérez-Jiménez, L. Valencia-Cabrera, B. Wang, X. Zeng. Computing with viruses. *Theoretical Computer Science*, **623** (2016), 146-159.
3. Nigel J Dimmock, Andrew J Easton, and Keith Leppard. *Introduction to modern virology.* Blackwell Pub. Malden, MA (USA), 2007.
4. A. Romero-Jiménez, L. Valencia-Cabrera, M.J. Pérez-Jiménez. Generating Diophantine Sets by Virus Machines. In: Gong, M., Linqiang, P., Tao, S., Tang, K., Zhang, X. (eds) Bio-Inspired Computing – Theories and Applications. BIC-TA 2015. Communications in Computer and Information Science, vol 562. Springer, Berlin, Heidelberg.(2015)
5. A. Romero-Jiménez, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez. Computing partial recursive functions by Virus Machines. Lecture Notes in Computer Science, Volume 9504, 2015, pp 353-368 A preliminary version in J.M. Sempere and C. Zandron (eds.) Proceedings of the 16th International Conference on Membrane Computing (CMC16), 17-21 August, 2015, Valencia, Spain, pp. 307-321.
6. Gillespie, D. T. (1991). Markov processes: an introduction for physical scientists. Elsevier.

[2] `http://www.gcn.us.es/19bwmc`

Author Index