Twentieth Brainstorming Week on Membrane Computing

First International Workshop on Virus Machines

Sevilla, January 24 – 26, 2024

Francis George C. Cabarle
David Orellana-Martín
Gheorghe Păun
Agustín Riscos-Núñez
Editors

Twentieth Brainstorming Week on Membrane Computing

First International Workshop on Virus Machines

Sevilla, January 24 – 26, 2024

Francis George C. Cabarle
David Orellana-Martín
Gheorghe Păun
Agustín Riscos-Núñez
Editors

RGNC REPORT 1/2024

Research Group on Natural Computing
Universidad de Sevilla

Sevilla, 2024

ISBN: 978-84-09-54839-2

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

Preface

The Twentieth Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from January 24 to 26, 2024, 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.

This edition has been celebrated in conjunction with the 1st International Workshop on Virus Machines, where this new model of computation was introduced to the P community and new ideas raised from the collaboration with the attendants of the conference.

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 2024 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. Henry N. Adorna, University of Philippines Diliman (Philippines) hnadorna@up.edu.ph
2. José A. Andreu Guzmán, Universidad de Sevilla (Spain), andreuguzman36@gmail.com
3. Rocco Ascone, University of Trieste (Italy) rocco.ascone@phd.units.it
4. Peter Battyányi, University of Debrecen (Hungary), battyanyi.peter@inf.unideb.hu
5. Giulia Bernardini, University of Trieste (Italy), giulia.bernardini@units.it
6. Francis George C. Cabarle, Universidad de Sevilla (Spain), fcabarle@us.es
7. Alberto d'Onofrio, University of Verona (Italy), ???
8. Giuditta Franco, University of Verona (Italy), giuditta.franco5@gmail.com
9. Rudolf Freund, TU Wien (Austria), rudi@emcc.at
10. Carmen Graciani, Universidad de Sevilla (Spain), cgdiaz@us.es
11. Anna Kuczik University of Debrecen (Hungary), kuczik.anna@inf.unideb.hu
12. Alberto Leporati University of Milan–Bicocca (Italy) alberto.leporati@unimib.it
13. Luca Manzoni, University of Trieste (Italy), lmanzoni@units.it
14. Fareed Muhammad Mazhar, University of Milano-Bicocca (Italy), muhammadmazhar.fareed@studenti.univr.it
15. David Orellana-Martín, Universidad de Sevilla (Spain), dorellana@us.es
16. Andrei Păun, Universidad de Sevilla (Spain), andreipaun@gmail.com
17. Mario J. Pérez-Jiménez, Universidad de Sevilla (Spain), marper@us.es
18. Prithwineel Paul, University of Engineering and Management (India), prithwineel@iem.edu.in
19. Antonio Ramírez-de-Arellano, Universidad de Sevilla (Spain), aramirezdearellano@us.es
20. Agustín Riscos-Núñez, Universidad de Sevilla (Spain), ariscosn@us.es
21. Álvaro Romero-Jiménez, Universidad de Sevilla (Spain), romero.alvaro@us.es
22. José M. Sempere, Universitat Politècnica de València (Spain), jsempere@dsic.upv.es
23. Bosheng Song, Hunan University (China), boshengsong@hnu.edu.cn
24. Luis Valencia-Cabrera, Universidad de Sevilla (Spain), lvalencia@us.es
25. Davide Valcamonica, University of Milano-Bicocca (Italy),
26. György Vaszil, University of Debrecen (Hungary), vaszil.gyorgy@inf.unideb.hu
27. Tao Wang, Xihua University (China), wangatao2005@163.com
28. XiangXiang Zeng, Hunan University (China), xzeng@hnu.edu.cn
29. Gexiang Zhang, Chengdu University of Information Technology (China), zhgxdylan@126.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) VII Plan Propio, *Vicerrectorado de Investigación de la Universidad de Sevilla*, (ii) Department of Computer Science and Artificial Intelligence from *Universidad de Sevilla*, (iii) the SCORE Lab of the *Universidad de Sevilla* (iv) the Research Institute of Computer Engineering (I3US) of the *Universidad de Sevilla*, and (v) the BioBit project from *Zhejiang Lab*.

The Editors
(Sep 2024)

Contents

Global Attractors of Reactantless and Inhibitorless Reaction Systems

Rocco Ascone, Giulia Bernardini, Luca Manzoni

[1]University of Trieste
E-mail: rocco.ascone@phd.units.it, giulia.bernardini@units.it, lmanzoni@units.it

Summary. In this study, we explore the computational complexity of deciding the existence of fixed points and cycles that can be reached from any other states (also called *global attractors*) in the dynamics of inhibitorless and reactantless reaction systems. The same problems were proved to be **PSPACE**-complete in the case of unconstrained reaction systems. We show, in contrast, that in the considered resource-bounded classes deciding whether a global fixed point attractor exists can be done in polynomial time. Furthermore, we prove that only trivial cycles consisting of a single state can exist in the dynamics of inhibitorless systems, while in reactantless systems cycles of two states may occur, and it is **coNP**-hard to decide their existence.

1 Introduction

Introduced nearly two decades ago by Ehrenfeucht and Rozenberg [1], reaction systems are an abstract computational model inspired by the chemical reactions occurring in living cells. The notion at the heart of this model is that the biochemical processes within a cell can be simulated using a limited collection of entities that represent various substances, alongside a set of rules that mimic reactions. A reaction is characterized by its reactants, inhibitors, and products, and it occurs when the set of entities currently present in the cell (ie the system's state) includes all reactants and lacks any inhibitors, resulting in the reaction's products.

Whenever a set of reactions takes place in a certain state, the system's subsequent state is determined by the union of the products of all the occurred reactions. This process defines a dynamical system whose points are given by all the possible subsets of entities, ie all possible states of the reaction system. Determining the computational complexity of deciding on the occurrence of various behaviours of such dynamical systems has been the object of a great deal of research work [2, 3, 4, 5, 6, 7, 8].

Reaction systems operate on a qualitative basis, meaning that the presence of a reactant in a given state implies it is available in sufficient quantities for all reactions that require it, thus avoiding any conflicts over shared resources. Other

related models have been proposed in the literature that waive this assumption, see eg [9, 10, 11, 12, 13]. Nevertheless, the computational power of the simpler qualitative model has been demonstrated by several studies [14, 15, 16, 17, 18] showing that reaction systems can be effectively used to simulate various biological processes.

Although the conventional framework for reaction systems does not limit the number of reactants and inhibitors involved in each reaction, an alternative branch of research concentrates on systems with constrained resources. Ehrenfeucht et al. [19] first investigated how bounding the number of reactants and inhibitors in the reactions can affect the kinds of functions that a reaction system can define. Manzoni et al. [20] then classified resource-bounded systems in such a way that the reaction functions enjoy specific properties within each class: in particular, they identified the class of *inhibitorless* reaction systems, in which all reactions have an empty set of inhibitors; the class of *reactantless* systems in which the set of reactants is always empty; and the class of minimal-resources systems, later named *additive* [21], in which each reaction only uses one reactant and no inhibitors.

Dennunzio et al. [22] studied the complexity of reachability in several subclasses of inhibitorless and reactantless systems; Azimi et al. [23] studied how to list all steady states of a system whose reactions have a small quantity of both reactants and inhibitors; and Ascone et al. investigated the computational complexity of problems related to the existence of fixed points and attractors in reactantless and inhibitorless systems [24] and in additive systems [21].

Contributions.

In this paper, we study the computational complexity of deciding on the existence of fixed points and cycles that are also *global attractors* (ie they can be reached from every other state) in inhibitorless and reactantless reaction system. All these problems were shown to be **PSPACE**-complete in unconstrained reaction systems [25]: in contrast, we show that disabling either the set of reactants or the set of inhibitors reduces to polynomial the complexity of deciding whether a global fixed point attractor exists, as well as determining if a given state is a global attractor. Furthermore, we prove that only trivial cycles consisting of a single state can exist in the dynamics of inhibitorless systems, while in reactantless systems cycles of two states may occur, and it is **coNP**-hard to decide on their existence. Table 1 summarises our results.

2 Basics Notions

Given a finite set S of *entities*, a *reaction* a over S is a triple (R_a, I_a, P_a) of subsets of S; we call R_a the set of *reactants*, I_a the set of *inhibitors*, and P_a the nonempty set of *products*. Note that, in this paper, the reactants and inhibitors of a reaction are allowed to be empty sets as in the original definition of reaction systems [1]. The set of all reactions over S is denoted by $\mathrm{rac}(S)$. A *reaction system* (RS) $\mathcal{A} = (S, A)$

Problem		$\mathcal{RS}(\infty,\infty)$	$\mathcal{RS}(0,\infty)$	$\mathcal{RS}(\infty,0)$
A given state is a global attractor		**PSPACE**-c [25]	**P** (Cor. 13)	**P** (Cor. 5)
$\exists$ global fixed point attractor		**PSPACE**-c [25]	**P** (Cor. 14)	**P** (Cor. 6)
$\exists$ global cycle attractor of length at least k	$k=2$	**PSPACE**-c [25]	**coNP**-hard (Thm. 18)	$\nexists$ (Lemma 7)
	$k>2$	**PSPACE**-c [25]	$\nexists$ (Pro. 15)	$\nexists$ (Lemma 7)

Table 1: Computational complexity of the problems studied in this work for different classes of reaction systems. $\mathcal{RS}(\infty,\infty)$, $\mathcal{RS}(0,\infty)$ and $\mathcal{RS}(\infty,0)$ denote unconstrained, reactantless and inhibitorless reaction systems, respectively (see Def. 1). Light-blue cells contain the results proved in this paper.

where S consists of the finite set of entities S, called the *background set*, and a set $A \subseteq \mathrm{rac}(S)$ of reactions over S.

We call any subset of S a *state* of the reaction system; a reaction a is *enabled* in a state T when $R_a \subseteq T$ and $I_a \cap T = \varnothing$, and the set of all the reactions from $\mathcal{A}$ enabled in T is denoted by $\mathrm{en}_{\mathcal{A}}(T)$. The *result function* $\mathrm{res}_a : 2^S \to 2^S$ of a reaction a, where 2^S denotes the power set of S, is defined as

$$\mathrm{res}_a(T) \coloneqq \begin{cases} P_a & \text{if } a \text{ is enabled in } T \\ \varnothing & \text{otherwise.} \end{cases}$$

The definition of res_a naturally extends to sets of reactions: given any $T \subseteq S$ and $A \subseteq \mathrm{rac}(S)$, we define $\mathrm{res}_A(T) \coloneqq \bigcup_{a\in A} \mathrm{res}_a(T)$. Consistently, the result function $\mathrm{res}_{\mathcal{A}}$ of the whole RS $\mathcal{A} = (S, A)$ is defined as equal to res_A, i.e., the result function of the whole set of reactions of the reaction system. In this way, any RS $\mathcal{A} = (S, A)$ induces a discrete dynamical system with state set 2^S and next state function $\mathrm{res}_{\mathcal{A}}$.

In this paper, we are interested in the dynamics of RS, i.e., the study of the successive states of the system under the action of the result function $\mathrm{res}_{\mathcal{A}}$ starting from some initial set of entities. The *orbit* or *state sequence* of a given state T of a RS $\mathcal{A}$ is defined as the sequence of states obtained by subsequent iterations of $\mathrm{res}_{\mathcal{A}}$ starting from T, namely the sequence $(T, \mathrm{res}_{\mathcal{A}}(T), \mathrm{res}^2_{\mathcal{A}}(T), \dots)$. Note that since S is finite, for any state T the sequence $(\mathrm{res}^n_{\mathcal{A}}(T))_{n\in\mathbb{N}}$ is always ultimately periodic. In particular, the orbit of a state T is a *cycle* of length k if there exists $k \in \mathbb{N}$ such that $\mathrm{res}^k_{\mathcal{A}}(T) = T$, and $\mathrm{res}^h_{\mathcal{A}}(T) \neq T$ for every $h < k$. In the special case where $k = 1$, T is said to be a *fixed point*.

Any set of cycles forms an *invariant set* for $\mathcal{A}$, that is, a set of states $\mathcal{U} \subseteq 2^S$ such that $\cup_{U\in\mathcal{U}}\{\mathrm{res}_{\mathcal{A}}(U)\} = \mathcal{U}$. In particular, it is also true that any invariant set for $\mathcal{A}$ is a set of cycles [25]. A *local attractor* for $\mathcal{A}$ is an invariant set $\mathcal{U}$ such that there exists an external state $T \notin \mathcal{U}$ such that $\mathrm{res}_{\mathcal{A}}(T) \in \mathcal{U}$. An invariant set $\mathcal{U}$ is a *global attractor* if for all states $T \in 2^S$ there exists $k \in \mathbb{N}$ such that $\mathrm{res}^k_{\mathcal{A}}(T) \in \mathcal{U}$, i.e., $\mathcal{U}$ is eventually reached from every possible state of $\mathcal{A}$. When a global attractor $\mathcal{U}$ consists of only one state T, we say that T is a *global fixed-point attractor*. Similarly, $\mathcal{U}$ is a *global cycle attractor* if all the states in $\mathcal{U}$ belong to the same cycle.

We now recall the classification of reaction systems in terms of the number of resources employed per reaction [20].

Definition 1 ([20])**.** Let $i, r \in \mathbb{N}$. The class $\mathcal{RS}(r, i)$ consists of all RS having at most r reactants and i inhibitors for reaction. We also define the (partially) unbounded classes $\mathcal{RS}(\infty, i) = \bigcup_{r=0}^{\infty} \mathcal{RS}(r, i)$, $\mathcal{RS}(r, \infty) = \bigcup_{i=0}^{\infty} \mathcal{RS}(r, i)$, and $\mathcal{RS}(\infty, \infty) = \bigcup_{r=0}^{\infty} \bigcup_{i=0}^{\infty} \mathcal{RS}(r, i)$.

We will call $\mathcal{RS}(0, \infty)$ the class of *reactantless* systems, and $\mathcal{RS}(\infty, 0)$ the class of *inhibitorless* systems.

Note that the classification of Definition 1 does not consider the number of products as a parameter because RS can always be assumed to be in *singleton product normal form* [26]: any reaction $(R, I, \{p_1, \ldots, p_m\})$ can be replaced by the set of reactions $(R, I, \{p_1\}), \ldots, (R, I, \{p_m\})$ which produce the same result.

Five equivalence classes of RS implied by Definition 1 have a characterisation in terms of functions over the Boolean lattice 2^S [20], listed in Table 2. Recall that

Class of RS	**Subclass of** $2^S \to 2^S$
$\mathcal{RS}(\infty, \infty)$	all
$\mathcal{RS}(0, \infty)$	antitone
$\mathcal{RS}(\infty, 0)$	monotone
$\mathcal{RS}(1, 0)$	additive
$\mathcal{RS}(0, 0)$	constant

Table 2: Functions computed by several classes of RS.

a function $f : 2^S \to 2^S$ is *antitone* if $X \subseteq Y$ implies $f(X) \supseteq f(Y)$, *monotone* if $X \subseteq Y$ implies $f(X) \subseteq f(Y)$, *additive* (or an upper-semilattice endomorphism) if $f(X \cup Y) = f(X) \cup f(Y)$ for all $X, Y \in 2^S$. We say that the RS $\mathcal{A} = (S, A)$ computes the function $f : 2^S \to 2^S$ if $\mathrm{res}_{\mathcal{A}} = f$.

3 Global Attractors of Inhibitorless RS

In this section, we study the complexity of deciding the existence of a global fixed-point attractor or a global cycle attractor in inhibitorless reaction systems.

3.1 Existence of a global fixed-point attractor

We begin with a simple observation which follows immediately from the definition of global fixed-point attractors.

Observation 2. A reaction system with a global fixed-point attractor cannot have any other fixed points or cycles.

In particular, Observation 2 implies that if a global fixed-point attractor exists, it is unique. Proposition 3 provides a characterization of global fixed-point attractors for monotone functions.

Proposition 3. *Let S be a finite set of n elements, $f : 2^S \to 2^S$ a monotone function and T a fixed point for f consisting of t elements. Then, T is a global fixed-point attractor for f if and only if $f^t(\varnothing) = T = f^{n-t}(S)$.*

Proof. $\boxed{\Rightarrow}$ *Consider the sequence $\varnothing \subsetneq f(\varnothing) \subsetneq \cdots \subsetneq f^m(\varnothing) = f^{m+1}(\varnothing)$. If it was $f^m(\varnothing) \subsetneq T$, there would exist a fixed point different from T, therefore T would not be a global attractor by Observation 2. Thus it must be $f^m(\varnothing) = T$ and since $|T| = t$, because of the monotonicity of f, it must also be $m \leq t$, implying that $f^t(\varnothing) = T$. Consider now the sequence $S \supsetneq f(S) \supsetneq \cdots \supsetneq f^k(S) = f^{k+1}(S)$. If it was $f^k(S) \supsetneq T$, then there would exist a fixed point different from T, therefore T would not be a global attractor. We obtain that $f^k(S) = T$; since $t = |T| = |f^k(S)| \leq n - k$ then by monotonicity it must be $k \leq n - t$, thus $f^{n-t}(S) = T$.*

$\boxed{\Leftarrow}$ *We need to prove that $T = f^t(\varnothing) = f^{n-t}(S)$ is a global attractor. Consider any state $\varnothing \subsetneq T' \subsetneq S$: by monotonicity, it holds that $T = f^t(\varnothing) \subseteq f^t(T')$ and $f^{n-t}(T') \subseteq f^{n-t}(S) = T$. We divide two cases.*
Case (i): $t \leq n-t$. Since $T \subseteq f^t(T')$, then it holds $f^{n-2t}(T) \subseteq f^{n-2t+t}(T') \Rightarrow T \subseteq f^{n-t}(T') \subseteq T$, and therefore T' reaches T in at most $n - t$ steps.
Case (ii): $t > n - t$. Since $T \supseteq f^{n-t}(T')$, then $f^{2t-n}(T) \supseteq f^{2t-n+n-t}(T') \Rightarrow T \supseteq f^t(T') \supseteq T$, therefore T' reaches T in at most t steps.

Proposition 3 thus immediately gives a criterion for deciding the existence of a global fixed-point attractor for monotone functions.

Corollary 4. *Given S a finite set of n elements, and $f : 2^S \to 2^S$ monotone, there exists a global fixed-point attractor if and only if $f^t(\varnothing) = f^{n-t}(S)$ for some $0 \leq t \leq n$.*

Proposition 3 and Corollary 4 can be directly applied to inhibitorless reaction systems, whose result functions are always monotone. We obtain the following results.

Corollary 5. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and a state $T \subseteq S$, deciding if T is a global fixed-point attractor of $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_\mathcal{A}$ is monotone [20], we can apply Proposition 3. Therefore, T is a global attractor for $\mathcal{A}$ if and only if $\mathrm{res}^t_\mathcal{A}(\varnothing) = T = \mathrm{res}^{n-t}_\mathcal{A}(S)$ where t and n are the cardinalities of T and S, respectively. For any state $U \subseteq S$, $\mathrm{res}_\mathcal{A}(U)$ can be computed in polynomial time: it suffices to check which reactions are enabled in U by intersecting their reactants and inhibitors with U, and then take the union of the products of the enabled functions. To decide if T is a global attractor we only need to evaluate $\mathrm{res}_\mathcal{A}$ at most $|S|$ times, thus the problem is in **P**.

Corollary 6. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and a state $T \subseteq S$, deciding on the existence of a global fixed-point attractor for $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_{\mathcal{A}}$ *is monotone [20], we can apply Corollary 4. Therefore, there exists a global attractor for* $\mathcal{A}$ *if and only if* $\mathrm{res}^t_{\mathcal{A}}(\varnothing) = \mathrm{res}^{n-t}_{\mathcal{A}}(S)$ *for some* $0 \leq t \leq n$ *where* n *is the cardinality of* S*. We conclude as in Corollary 5.*

3.2 Existence of a global cycle attractor

We begin this section with a result that immediately follows from the Knaster-Tarki theorem [27] and excludes the existence of a global cycle attractor of length greater than one in the case of monotone functions. In particular, this implies that no global cycle attractor of length $k \geq 2$ can exist in the dynamics of inhibitorless reaction systems, as their result function is always monotone [20].

Lemma 7. *Let* $f : 2^S \to 2^S$ *be a monotone function. Then no global attractor* k*-cycle exists for any* $k \geq 2$*. Moreover, if* $\mathcal{U}$ *is a global attractor invariant set, then at least one of the cycles in* $\mathcal{U}$ *is a fixed point.*

Proof. By the Knaster-Tarki theorem, monotone functions always have a fixed point, therefore a global attractor k-cycle cannot exist for $k > 1$ by Obsservation 2. For the same reason, if $\mathcal{U}$ is a global attractor invariant set, then at least one of the cycles in $\mathcal{U}$ is a fixed point.

The rest of this section provides results on the existence of global attractors consisting of two fixed points for monotone functions (thus for inhibitorless reaction systems). These results will be useful in Section 4 to prove the complexity of deciding on the existence of global cycle attractors in *reactantless* systems. In Lemma 8, we prove that for any monotone function, a global attractor consisting of two fixed points must have a particular form.

Lemma 8. *Let* $f : 2^S \to 2^S$ *monotone and* $\mathcal{U} = \{T_1, T_2\}$ *a global attractor consisting of two fixed points, then* $\mathcal{U} = \{f^n(\varnothing), f^m(S)\}$*, with* $n, m \geq 0$ *such that* $f^n(\varnothing) = f^{n+1}(\varnothing)$ *and* $f^m(S) = f^{m+1}(S)$*.*

Proof. By monotonicity, $f^n(\varnothing) \subseteq T_i \subseteq f^m(S)$ *for* $i = 1, 2$*. Suppose for a contradiction that the inclusions are both strict: then* $\mathcal{U}$ *would not be a global attractor by Observation 2, a contradiction. We obtain the statement.*

Lemma 8 implies that any global attractor consisting of two fixed points in a reaction system $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ must be of the form $\{\mathrm{res}^n_{\mathcal{A}}(\varnothing), \mathrm{res}^m_{\mathcal{A}}(S)\}$. However, this characterization is not strong enough to give a polynomial time algorithm, and in Proposition 9 we prove that deciding if $\mathrm{res}^n_{\mathcal{A}}(\varnothing)$ and $\mathrm{res}^m_{\mathcal{A}}(S)$ are the only fixed points for $\mathcal{A}$ is **coNP**-complete. The proof extends an idea from [24, Theorem 25].

Proposition 9. *Given* $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ *such that* $\varnothing$ *and* S *are fixed points, it is* **coNP***-complete to decide if* $\varnothing$ *and* S *are the only fixed points.*

Proof. The problem lies in **coNP** *because there exists a simple non-deterministic algorithm which guesses a state* T *and then verifies in polynomial time that it is a fixed point different from* $\varnothing$ *and* S*. To show* **coNP***-completeness, we reduce*

VALIDITY *[28] to this problem. Given a Boolean formula* $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$ *in DNF over the variables* $V = \{x_1, \ldots, x_n\}$*, let* $\overline{V} \coloneqq \{\overline{x_j} : x_j \in V\}$ *and* $\heartsuit_S \coloneqq \{\heartsuit_i : 1 \leq i \leq n\}$*. We define* $pos(\varphi_r) \subseteq V$ *the set of variables that occur non-negated in* φ_r *and* $\overline{neg}(\varphi_r) \subseteq \overline{V}$ *the set of variables that occur negated in* φ_r*. We then define a RS* $\mathcal{A}$ *with background set* $S \coloneqq V \cup \overline{V} \cup \heartsuit_S \cup \{\diamondsuit\}$ *and reactions*

$$(\overline{neg}(\varphi_j) \cup pos(\varphi_j) \cup \heartsuit_S, \varnothing, \{\diamondsuit\}) \qquad \text{for } 1 \leq j \leq m \tag{1}$$
$$(\{x_i\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, x_i\}) \qquad \text{for } 1 \leq i \leq n \tag{2}$$
$$(\{\overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, \overline{x_i}\}) \qquad \text{for } 1 \leq i \leq n \tag{3}$$
$$(\{x_i, \overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\diamondsuit\}) \qquad \text{for } 1 \leq i \leq n \tag{4}$$
$$(\{\diamondsuit\} \cup \heartsuit_S, \varnothing, S). \tag{5}$$

Note that $\varnothing$ *and* S *are fixed points; furthermore, any* $T \subseteq S$*, it falls in one of the following cases:*

1) $\heartsuit_S \nsubseteq T$*. In this case,* $\mathrm{res}_{\mathcal{A}}(T) = \varnothing$*, since no reaction is enabled;*
2) $\diamondsuit \in T$ *and* $\heartsuit_S \subseteq T$*. In this case, reaction (5) is enabled and thus* $\mathrm{res}_{\mathcal{A}}(T) = S$*;*
3) T *is of the form* $Y \cup \heartsuit_S$*, with* $Y \subseteq V \cup \overline{V}$*.*

Thus $\varnothing$ *is reached from any state that does not fully contain* $\heartsuit_S$*, and* S *from any state containing both* $\heartsuit_S$ *and* $\diamondsuit$*. Let us now focus on the states falling in case (3). For any* $Y \subseteq V \cup \overline{V}$*, we define* $\heartsuit_Y \coloneqq \{\heartsuit_i : x_i \in Y \vee \overline{x_i} \in Y\} \subseteq \heartsuit_S$*. The following subcases can happen:*

3.1) $\exists\, i$ *such that both* $x_i, \overline{x_i} \in Y$*. In this case, the* i*-th reaction of group (4) is enabled by* $Y \cup \heartsuit_S$*, thus* $\diamondsuit \in \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$*; if* $\heartsuit_S \subseteq \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$ *or* $\heartsuit_S \nsubseteq \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$*, then* $\mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$ *is either in case (1) or (2) above, implying that* $\mathrm{res}^2_{\mathcal{A}}(Y \cup \heartsuit_S) \in \{S, \varnothing\}$*;*
3.2) $\exists\, i$ *such that both* $x_i, \overline{x_i} \notin Y$*. Then* $\heartsuit_S \nsubseteq \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$ *since none of the* i*-th reactions in groups (2), (3) are enabled, therefore* $\mathrm{res}^2(Y \cup \heartsuit_S) = \varnothing$*.*
3.3) $x_i \in Y \Leftrightarrow \overline{x_i} \notin Y$ *for every* $1 \leq i \leq n$*. In this case,* $Y = X \cup \overline{V \setminus X}$ *for some* $X \subseteq V$*, thus it encodes an assignment for* φ *where the variables in* X *are assigned true value and the variables in* $V \setminus X$ *are assigned value false. Note that being* φ *in DNF, it is satisfied if and only if at least one* φ_i *is satisfied; moreover, any clause* φ_i*, being a conjunction of variables, is satisfied if and only if all of its negated variables are assigned value false and all of its non-negated variables are assigned value true. Therefore, the assignment implied by* $X \cup \overline{V \setminus X}$ *satisfies* φ *if and only if* $X \cup \overline{V \setminus X} \cup \heartsuit_S$ *enables one of the reactions from the group (1). Hence, if* $Y = X \cup \overline{V \setminus X}$ *satisfies* φ *then* $\diamondsuit \in \mathrm{res}(Y \cup \heartsuit_S)$*, implying* $\mathrm{res}^2(Y \cup \heartsuit_S) = S$*. If instead* Y *does not satisfy* φ *then* $\mathrm{res}(Y \cup \heartsuit_S) = Y \cup \heartsuit_S$ *by reactions of groups (2) and (3).*

We conclude that $\mathcal{A}$ *has no fixed points other than* $\varnothing$ *and* S *if and only if all the assignments satisfy* φ*, ie* φ *is a tautology. Since the mapping* $\varphi \mapsto \mathcal{A}$ *is computable in polynomial time, the problem is* **coNP***-hard.*

Since a necessary condition for $\mathcal{U} = \{\varnothing, S\}$ to be a global attractor is that $\varnothing$ and S are the only two fixed points, Proposition 9 has the following immediate corollary.

Corollary 10. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points, it is* **coNP***-hard to decide if $\mathcal{U} = \{\varnothing, S\}$ is a global attractor.*

4 Global Attractors of Reactantless RS

4.1 Existence of a global fixed-point attractor

We begin this section with a characterization of global fixed-point attractors when the function is antitone. Corollary 12, analogously to Corollary 4 for the monotone case, will then provide a criterion for deciding the existence of a global fixed-point attractor for antitone functions in polynomial time.

Proposition 11. *Let S be a finite set, $f : 2^S \to 2^S$ antitone and T a fixed point for f. Then, T is global fixed-point attractor for f if and only if T is a global fixed-point attractor for f^2.*

Proof. $\boxed{\Rightarrow}$ *Since T is a fixed point for f, it is also a fixed point for f^2. We need to prove that T is a global attractor for f^2, but since for every state $T' \subseteq S$ there exists $t \in \mathbb{N}$ such that $f^t(T') = T$, then $(f^2)^t(T') = f^{2t}(T') = f^2(T) = T$.*

$\boxed{\Leftarrow}$ *Consider T a global fixed-point attractor for f^2. Then it must hold that $f(T) = T$, as otherwise, $f(T) \neq T$ would imply that $f^2(f(T)) = f(T)$ and thus $f(T)$ would be a fixed point for f^2 different from T, which is a contradiction by Observation 2. $f(T) = T$ implies that T is also a global fixed-point attractor for f, because for every $T' \subseteq S$, T' reaches T in t steps through f^2, thus T' reaches T in $2t$ steps through f.*

Corollary 12. *Given S a finite set and $f : 2^S \to 2^S$ antitone, a global fixed-point attractor for f exists if and only if there exists a global fixed-point attractor for f^2.*

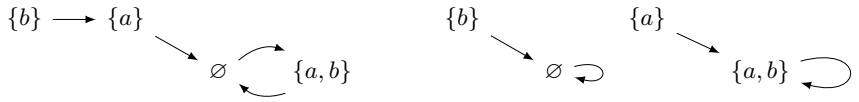
Proposition 11 and Corollary 12 can be straightforwardly applied to result functions of reactantless reaction systems, leading to the following two results.

Corollary 13. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$ and a state $T \subseteq S$, deciding if T is a global fixed-point attractor of $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_\mathcal{A}$ is antitone [20], Proposition 11 applies. Therefore, T is a global attractor for $\mathcal{A}$ if and only if T is a global fixed-point attractor for $\mathrm{res}^2_\mathcal{A}$. Since $\mathrm{res}^2_\mathcal{A}$ is monotone, we can proceed as in the proof of Corollary 5, and decide whether T is a global attractor simply by evaluating $\mathrm{res}_\mathcal{A}$ at most $2|S|$ times.

Corollary 14. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$ and a state $T \subseteq S$, deciding whether there exists a global fixed-point attractor of $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_\mathcal{A}$ is antitone [20], Corollary 12 applies, implying that there exists a global fixed-point attractor for $\mathrm{res}_\mathcal{A}$ if and only if there exists a global fixed-point attractor for $\mathrm{res}^2_\mathcal{A}$. We conclude as in Corollary 13.



(a) A global 2-cycle attractor.

(b) A global attractor consisting of two fixed points.

Fig. 1: Representation of the dynamics of Example 16.

4.2 Existence of a global cycle attractor

We begin this section by showing, in Proposition 15, that a global k-cycle attractor cannot exist for any antitone function for any $k > 2$: see also Example 16.

Proposition 15. *Let $\mathcal{U}$ be a global cycle attractor for an antitone function $f : 2^S \to 2^S$, then there exists $T \subseteq S$ such that either $\mathcal{U} = \{T\}$ or $\mathcal{U} = \{T, f(T)\}$.*

Proof. Let $f^2(\mathcal{U}) := \{f^2(U) : U \in \mathcal{U}\}$; this is a global attractor invariant set for f^2. Suppose $\mathcal{U}$ is a $(2k+1)$-cycle for some $k \geq 0$: then $f^2(\mathcal{U})$ is also a $(2k+1)$-cycle. Since by Lemma 7 every global attractor invariant set for a monotone function must contain a fixed point, and since f being antitone implies f^2 being monotone, it must be $k = 0$ and thus $\mathcal{U} = f^2(\mathcal{U}) = \{T\}$ must be a global fixed-point attractor for f^2. Suppose now $\mathcal{U}$ is a $(2k)$-cycle for some $k \geq 1$: then $f^2(\mathcal{U})$ consists two k-cycles. Since one of the two cycles must be a fixed point by Lemma 7, it must be $k = 1$ and thus $\mathcal{U} = \{T, f(T)\}$ for some $T \subseteq S$.

Example 16. Let $S = \{a, b\}$ and $f : 2^S \to 2^S$ given by:

$$f(\varnothing) = \{a, b\}; \quad f(\{a\}) = \varnothing; \quad f(\{b\}) = \{a\}; \quad f(\{a, b\}) = \varnothing.$$

f is clearly antitone and in the dynamics, we have a global 2-cycle attractor $\{\varnothing, S\}$: see Figure 1a. Consider now $f^2 : 2^S \to 2^S$, given by

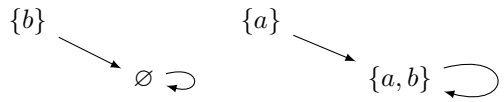
$$f^2(\varnothing) = \varnothing; \quad f^2(\{a\}) = \{a, b\}; \quad f^2(\{b\}) = \varnothing; \quad f^2(\{a, b\}) = \{a, b\}.$$

f^2 has a global attractor consisting of two fixed points, see Figure 1b. ⌟

From the proof of Proposition 15, we deduce that an antitone function $f : 2^S \to 2^S$ has a global 2-cycle attractor if and only if $f^2 : 2^S \to 2^S$ has a global attractor consisting of two fixed points.

The rest of this section is devoted to proving that deciding whether a reactantless RS has a 2-cycle global attractor reduces to the problem of Corollary 10 for inhibitorless systems, and it is, therefore, **coNP**-hard as well. We begin with an example that illustrates the workings of the reduction we will later provide in Theorem 18.

Example 17. Let $S = \{a, b\}$ and $\mathcal{A} = (S, A)$ an inhibitorless reaction system where $A = \{(\{a\}, \varnothing, \{a, b\})\}$. As already seen in Example 16, in the dynamics of $\mathcal{A}$ there are two fixed points that form together a global attractor (same dynamics as Figure 1b):



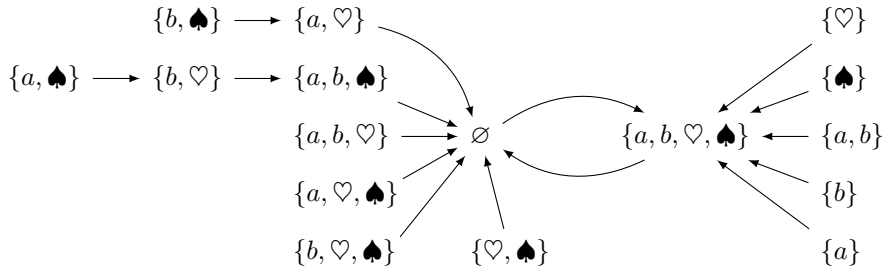
We want to construct a reactantless reaction system that can reproduce the dynamics of $\mathcal{A}$ for states $\varnothing \subsetneq T \subsetneq S$ and transform the global attractor of $\mathcal{A}$, consisting of two fixed points, into a 2-cycle global attractor. We thus construct $\mathcal{B} = (S', B)$ where $S' = \{a, b, \heartsuit, \spadesuit\}$ and B is given by the following reactions:

$$
\begin{aligned}
&(\varnothing, \{a, \heartsuit\}, \{a, \heartsuit\})\\
&(\varnothing, \{b, \heartsuit\}, \{b, \heartsuit\})\\
&(\varnothing, \{a, \spadesuit\}, \{a, b, \spadesuit\})\\
&(\varnothing, \{a, b, \heartsuit\}, \{a, b, \heartsuit, \spadesuit\})\\
&(\varnothing, \{a, b, \spadesuit\}, \{a, b, \heartsuit, \spadesuit\})\\
&(\varnothing, \{\heartsuit, \spadesuit\}, \{a, b, \heartsuit, \spadesuit\}).
\end{aligned}
$$

It is straightforward to verify that $\mathrm{res}_{\mathcal{B}}(\{b, \spadesuit\}) = \{a, \heartsuit\}$ and $\mathrm{res}_{\mathcal{B}}(\{a, \spadesuit\}) = \{b, \heartsuit\}$, thus

$$\mathrm{res}^2_{\mathcal{B}}(\{b, \spadesuit\}) = \varnothing \quad \text{and} \quad \mathrm{res}^2_{\mathcal{B}}(\{a, \spadesuit\}) = \{a, b, \spadesuit\}.$$

Note that in the original inhibitorless RS $\mathcal{A}$ we have $\mathrm{res}_{\mathcal{A}}(\{b\}) = \varnothing$ and $\mathrm{res}_{\mathcal{A}}(\{a\}) = \{a, b\}$, thus $\mathcal{B}$ can reproduce the dynamics of $\mathcal{A}$ in two steps starting from the states $\{a, \spadesuit\}$ and $\{b, \spadesuit\}$ and going through the states the states $\{a, \heartsuit\}$ and $\{b, \heartsuit\}$. The last three reactions of B ensure that there is a 2-cycle global attractor, as all the states except for $\{a, \spadesuit\}$, $\{b, \spadesuit\}$, and $\{b, \heartsuit\}$ reach the 2-cycle $\{\varnothing, S'\}$ in one step, which makes it a global 2-cycle attractor. The dynamics of $\mathcal{B}$ is the following:



In Theorem 18, we extend and generalize the construction of Example 17 to any $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ to reduce the problem of deciding whether $\mathcal{U} = \{\varnothing, S\}$ is a global attractor in inhibitorless reaction systems to the problem of deciding the existence of a global 2-cycle attractor in reactantless reaction systems.

Theorem 18. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$, deciding if there exists a 2-cycle global attractor is* **coNP***-hard.*

Proof. We reduce from the problem of deciding if $\mathcal{U} = \{\varnothing, S\}$ *is a global attractor in inhibitorless reaction systems (see Corollary 10). More precisely, given* $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ *such that* $\varnothing$ *and* S *are fixed points, we want to construct in polynomial time a reaction system* $\mathcal{B} \in \mathcal{RS}(0, \infty)$ *such that* $\{\varnothing, S\}$ *is a global attractor for* $\mathcal{A}$ *if and only if there exists a 2-cycle global attractor for* $\mathcal{B}$*. We construct a reactantless RS* $\mathcal{B} \coloneqq (S', B)$*, with* $S' \coloneqq S \cup \{\heartsuit, \spadesuit\}$ *and* B *is given by the following reactions:*

$$(\varnothing, \{s, \heartsuit\}, \{s, \heartsuit\}) \qquad \text{for } s \in S \tag{6}$$
$$(\varnothing, R_a \cup \{\spadesuit\}, P_a \cup \{\spadesuit\}) \qquad \text{for } a = (R_a, \varnothing, P_a) \in A \tag{7}$$
$$(\varnothing, S \cup \{\heartsuit\}, S \cup \{\heartsuit, \spadesuit\}) \tag{8}$$
$$(\varnothing, S \cup \{\spadesuit\}, S \cup \{\heartsuit, \spadesuit\}) \tag{9}$$
$$(\varnothing, \{\heartsuit, \spadesuit\}, S \cup \{\heartsuit, \spadesuit\}). \tag{10}$$

Claim. All states of $\mathcal{B}$ *of the forms* $\{\spadesuit\}, \{\heartsuit\}$*,* $S \cup \{\heartsuit\}$*,* $S \cup \{\spadesuit\}$*,* T*, and* $T \cup \{\heartsuit, \spadesuit\}$*, for all* $T \subseteq S$*, reach* $\{\varnothing, S'\}$ *in one step. Furthermore,* $\operatorname{res}_{\mathcal{B}}(\varnothing) = S'$ *and* $\operatorname{res}_{\mathcal{B}}(S') = \varnothing$*.*

Proof. We immediately note that for any $T \subseteq S$ *we have* $\operatorname{res}_{\mathcal{B}}(T) = S \cup \{\heartsuit, \spadesuit\} = S'$ *since reaction (10) is enabled, and* $\operatorname{res}_{\mathcal{B}}(T \cup \{\heartsuit, \spadesuit\}) = \varnothing$ *since no reaction is enabled. By reactions (8) and (9), we also have* $\operatorname{res}_{\mathcal{B}}(\{\spadesuit\}) = \operatorname{res}_{\mathcal{B}}(\{\heartsuit\}) = S \cup \{\heartsuit, \spadesuit\} = S'$*. Furthermore, since* $\operatorname{res}_{\mathcal{A}}(\varnothing) = \varnothing$*, then* $R_a \neq \varnothing$ *for each* $a \in A$*, thus* $\operatorname{res}_{\mathcal{B}}(S \cup \{\heartsuit\}) = \varnothing$ *since no reaction is enabled, as well as* $\operatorname{res}_{\mathcal{B}}(S \cup \{\spadesuit\}) = \varnothing$*. Finally, since all the reactions are enabled by* $\varnothing$*, and no reaction is enabled by* $S' = S \cup \{\heartsuit, \spadesuit\}$*, we have that* $\operatorname{res}_{\mathcal{B}}(\varnothing) = S'$ *and* $\operatorname{res}_{\mathcal{B}}(S') = \varnothing$*. See also Figure 2 for a visual representation of the dynamics.*

Claim. The states $\{\spadesuit\}$*,* $\{\heartsuit\}$*,* $\{\heartsuit, \spadesuit\}$*,* S*,* $S \cup \{\heartsuit\}$*,* T*, and* $T \cup \{\heartsuit, \spadesuit\}$*, for all* $\varnothing \subsetneq T \subsetneq S$*, cannot be reached from any states.*

Proof. We can safely assume that in $\mathcal{A}$ *there are no reactions of the type* $(R_a, \varnothing, \varnothing)$*, because in any case, they do not affect the dynamic of* $\mathcal{A}$*. Therefore* $\operatorname{en}_{\mathcal{A}}(T) = \varnothing$ *if and only if* $\operatorname{res}_{\mathcal{A}}(T) = \varnothing$*. This implies that group (7) of the reactions of* $\mathcal{B}$ *does not contain any reactions of the form* $(\varnothing, R_a \cup \{\spadesuit\}, \{\spadesuit\})$*, implying that the state* $\{\spadesuit\}$ *cannot be reached from any state. With a similar reasoning we deduce that the states* $\{\heartsuit\}$*,* $\{\heartsuit, \spadesuit\}$*,* S*, and* T *for all* $\varnothing \subsetneq T \subsetneq S$ *cannot be reached from any state as well.*

Furthermore, none of the states of the form $T \cup \{\heartsuit, \spadesuit\}$ *with* $\varnothing \subsetneq T \subsetneq S$ *can be reached from any state: indeed, suppose for a contradiction that* $\operatorname{res}_{\mathcal{B}}(T') = T \cup \{\heartsuit, \spadesuit\}$ *for some* $T' \subseteq S'$ *and* $\varnothing \subsetneq T \subsetneq S$*. In order to obtain* $\heartsuit$ *in the product,* T' *must enable some reactions from group (6); and to obtain* $\spadesuit$*, it must also enable reactions from group (7). This implies* $\heartsuit, \spadesuit \notin T'$*, thus* $T' \subseteq S$ *and thus, by Claim 18,* $\operatorname{res}_{\mathcal{B}}(T') = S \cup \{\heartsuit, \spadesuit\}$*, which is a contradiction because by hypothesis* $T \subsetneq S$*. Finally,* $S \cup \{\heartsuit\}$ *cannot be reached from any state* U *because*

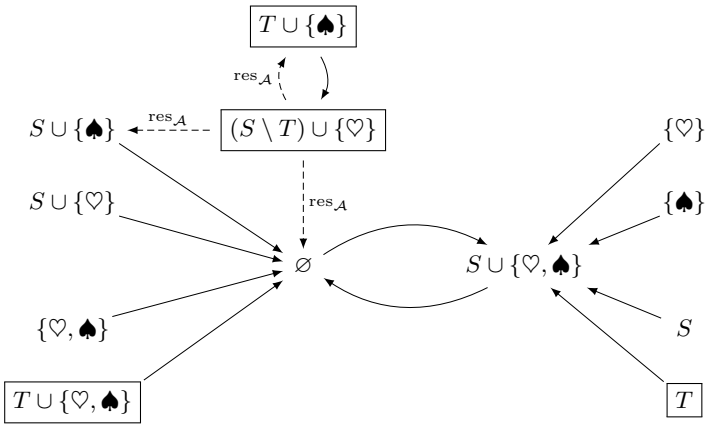


Fig. 2: Dynamics of the RS $\mathcal{B}$ in the reduction of Theorem 18. The states T, $T \cup \{\heartsuit, \spadesuit\}$, $T \cup \{\spadesuit\}$ and $(S \setminus T) \cup \{\heartsuit\}$ are a synthetic representation of the $2^S - 2$ states (one fore each $\varnothing \subsetneq T \subsetneq S$) of each type. The boxes around states $T \cup \{\spadesuit\}$ and $(S \setminus T) \cup \{\heartsuit\}$ hide the more refined dynamics for those states; dashed arcs represent the three existing possibilities for the dynamics of the states belonging to the boxes, as described after Claim 18.

this would require all and only the reactions from group (6) to be enabled in U, which can happen only if $U = \{\spadesuit\}$; but then reaction (8) is enabled as well, and indeed $\mathrm{res}_{\mathcal{B}}(\{\spadesuit\}) = S'$ *by Claim 18.*

It remains to determine the dynamics for the states of the form $T \cup \{\spadesuit\}$ and $T \cup \{\heartsuit\}$ for some $\varnothing \subsetneq T \subsetneq S$. Because of the reactions from group (6), we obtain

$$\mathrm{res}_{\mathcal{B}}(T \cup \{\spadesuit\}) = (S \setminus T) \cup \{\heartsuit\}; \tag{11}$$

and because of the reactions from group (7), in turn we have

$$\mathrm{res}_{\mathcal{B}}((S \setminus T) \cup \{\heartsuit\}) = \begin{cases} \mathrm{res}_{\mathcal{A}}(T) \cup \{\spadesuit\} & \text{if } \mathrm{en}_{\mathcal{A}}(T) \neq \varnothing \\ \varnothing & \text{otherwise,} \end{cases} \tag{12}$$

since $(S \setminus T) \cup \{\heartsuit\}$ enables $(\varnothing, R_a \cup \{\spadesuit\}, P_a \cup \{\spadesuit\})$ if and only if $(S \setminus T) \cap R_a = \varnothing$, which is the case if and only if $R_a \subseteq T$ and thus T enables $(R_a, \varnothing, P_a)$ in $\mathcal{A}$. As remarked in Claim 18, we have that $\mathrm{res}_{\mathcal{A}}(T) = \varnothing$ *if and only if* $\mathrm{en}_{\mathcal{A}}(T) = \varnothing$*, which is true if and only if* $\mathrm{res}_{\mathcal{B}}((S \setminus T) \cup \{\heartsuit\}) = \varnothing$*. We have obtained the following formula:*

$$\mathrm{res}^2_{\mathcal{B}}(T \cup \{\spadesuit\}) = \begin{cases} \mathrm{res}_{\mathcal{A}}(T) \cup \{\spadesuit\} & \text{if } \mathrm{en}_{\mathcal{A}}(T) \neq \varnothing \\ \varnothing & \text{otherwise.} \end{cases} \tag{13}$$

Therefore, iterating (13), if $\mathrm{res}_{\mathcal{A}}^{i}(T) \notin \{\varnothing, S\}$ *for all* $i = 1, \ldots, k$*, we obtain*

$$\mathrm{res}_{\mathcal{B}}^{2k}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{A}}^{k}(T) \cup \{\spadesuit\}\,. \tag{14}$$

Note that the states of the form $T \cup \{\heartsuit\}$ *with* $\varnothing \subsetneq T \subsetneq S$ *coincide with the states of the form* $(S \setminus T) \cup \{\heartsuit\}$*; in particular, any such state* $T \cup \{\heartsuit\}$ *is reached from* $(S \setminus T) \cup \{\spadesuit\}$ *by Equation (11), and reaches either* $\varnothing$ *or* $\mathrm{res}_{\mathcal{A}}(S \setminus T) \cup \{\spadesuit\}$ *according to Equation (12). In Figure 2, the states of the form* $T \cup \{\heartsuit\}$ *and* $T \cup \{\spadesuit\}$ *are compactly represented as boxed states, and their dynamics are not completely represented for the sake of readability.*

We observe that the only candidate 2*-cycle global attractor for* $\mathcal{B}$ *is* $\{\varnothing, S'\}$*, as it is a 2-cycle by Claim 18 and it is the only candidate global attractor by Claim 18 and the discussion below its proof. The next claim gives us the thesis.*

Claim. $\{\varnothing, S\}$ *is a global attractor for* $\mathcal{A}$ *if and only if* $\{\varnothing, S'\}$ *is a global attractor for* $\mathcal{B}$*.*

Proof. $\boxed{\Rightarrow}$ *Let* $\varnothing \subsetneq T \subsetneq S$*: in this case, we already proved in Claim 18 that* T *and* $T \cup \{\heartsuit, \spadesuit\}$ *reach* $\{\varnothing, S'\}$ *in one step. By hypothesis,* $\exists k \in \mathbb{N}$ *such that* $\mathrm{res}_{\mathcal{A}}^{k}(T) \in \{\varnothing, S\}$*. Let* k *be the minimum number that satisfies this property, implying that* $\mathrm{res}_{\mathcal{A}}^{i}(T) \notin \{\varnothing, S\}$ *for* $i = 1, \ldots, k-1$*. Thus we can apply Equation (14) and obtain*

$$\mathrm{res}_{\mathcal{B}}^{2(k-1)}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{A}}^{k-1}(T) \cup \{\spadesuit\}$$

Furthermore, applying Equation (11) to this result, we obtain

$$\mathrm{res}_{\mathcal{B}}^{2(k-1)+1}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}^{2k-1}(T \cup \{\spadesuit\}) = S \setminus \mathrm{res}_{\mathcal{A}}^{k-1}(T) \cup \{\heartsuit\}.$$

Since by hypothesis $\mathrm{res}_{\mathcal{A}}^{k}(T) \in \{\varnothing, S\}$*, there are two cases: if* $\mathrm{res}_{\mathcal{A}}^{k}(T) = S$*, then* $\mathrm{res}_{\mathcal{B}}^{2k}(T \cup \{\spadesuit\}) = S \cup \{\spadesuit\}$*, implying that* $\mathrm{res}_{\mathcal{B}}^{2k+1}(T \cup \{\spadesuit\}) = \varnothing$*. Otherwise,* $\mathrm{res}_{\mathcal{A}}^{k}(T) = \varnothing$*, which happens if and only if* $\mathrm{en}_{\mathcal{A}}(\mathrm{res}_{\mathcal{A}}^{k-1}(T)) = \varnothing$*: in this case,* $\mathrm{res}_{\mathcal{B}}^{2k}(T \cup \{\spadesuit\}) = \varnothing$ *by Equation (13). In any case,* $T \cup \{\spadesuit\}$ *reaches* $\{\varnothing, S'\}$ *in at most* $2k+1$ *steps. For the state* $T \cup \{\heartsuit\}$*, we can reduce to the previous case using Equation (12). Together with Claim 18, we obtain that if* $\{\varnothing, S\}$ *is a global attractor for* $\mathcal{A}$ *then* $\{\varnothing, S'\}$ *is a global attractor for* $\mathcal{B}$*.*

$\boxed{\Leftarrow}$ *Let* $\varnothing \subsetneq T \subsetneq S$*: by hypothesis, there exists* $k \in \mathbb{N}$ *such that* $\mathrm{res}_{\mathcal{B}}^{k}(T \cup \{\spadesuit\}) \in \{\varnothing, S'\}$*. Let* k *be the minimum number that satisfies that property. We want to prove that* T *is always attracted by* $\{S, \varnothing\}$*. We define two cases, depending on whether* k *is even or odd.*

1) $k = 2m$*. We have* $\mathrm{res}_{\mathcal{A}}^{i}(T) \notin \{\varnothing, S\}$ *for all* $i = 1, \ldots, m-1$ *as otherwise* $k = 2m$ *would not be the minimum. Thus we get* $\mathrm{res}_{\mathcal{B}}^{2m-1}(T \cup \{\spadesuit\}) = (S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T)) \cup \{\heartsuit\}$*. Since* $\varnothing \subsetneq \mathrm{res}_{\mathcal{A}}^{m-1}(T) \subsetneq S$*, we have* $\varnothing \subsetneq S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T) \subsetneq S$*, thus reaction (9) is not enabled by* $(S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T)) \cup \{\heartsuit\}$*, implying in turn that* $\heartsuit \notin \mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\})$*, and thus* $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) \neq S \cup \{\heartsuit, \spadesuit\}$*. But since* $\mathrm{res}_{\mathcal{B}}^{2m}(T) \in \{\varnothing, S'\}$ *then* $\mathrm{res}_{\mathcal{B}}^{2m}(T) = \varnothing$*. Suppose*

now for a contradiction that $\mathrm{res}^m_{\mathcal{A}}(T) \neq \varnothing$*: then it would also be* $\mathrm{res}^{2m}_{\mathcal{B}}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}((S \setminus \mathrm{res}^{m-1}_{\mathcal{A}}(T)) \cup \{\heartsuit\}) \neq \varnothing$*, a contradiction. We deduce that* $\mathrm{res}^m_{\mathcal{A}}(T) = \varnothing$*, thus* T *is attracted by* $\{S, \varnothing\}$*.*

2) $k = 2m+1$*. Clearly,* $\mathrm{res}^i_{\mathcal{A}}(T) \notin \{\varnothing, S\}$ *for* $i = 1, \ldots, m-1$*. Thus we have* $\mathrm{res}^{2m-1}_{\mathcal{B}}(T \cup \{\spadesuit\}) = (S \setminus \mathrm{res}^{m-1}_{\mathcal{A}}(T)) \cup \{\heartsuit\}$*. Since* $\mathrm{res}^{2m}_{\mathcal{B}}(T \cup \{\spadesuit\}) \neq \varnothing$ *then* $\mathrm{res}^m_{\mathcal{A}}(T) \neq \varnothing$*. Thus* $\mathrm{res}^{2m}_{\mathcal{B}}(T \cup \{\spadesuit\}) = \mathrm{res}^m_{\mathcal{A}}(T) \cup \{\spadesuit\}$*. Suppose for a contradiction that* $\mathrm{res}^m_{\mathcal{A}}(T) \subsetneq S$*, then* $\mathrm{res}^{2m+1}_{\mathcal{B}}(T \cup \{\spadesuit\}) = S \setminus \mathrm{res}^m_{\mathcal{A}}(T) \cup \{\heartsuit\} \notin \{\varnothing, S'\}$*, a contradiction by the definition of* k*. We deduce that* $\mathrm{res}^m_{\mathcal{A}}(T) = S$*, thus* T *is attracted by* $\{S, \varnothing\}$*.*

Summing up, we proved that if $\{\varnothing, S'\}$ *is a global attractor for* $\mathcal{B}$ *then* $\{\varnothing, S\}$ *is a global attractor for* $\mathcal{A}$*.*

Claim 18, together with Claim 18, directly implies that there exists 2*-cycle global attractor for* $\mathcal{B}$ *if and only if* $\{\varnothing, S\}$ *is a global attractor for* $\mathcal{A}$*. We also remark that the map* $\mathcal{A} \mapsto \mathcal{B}$ *can be constructed in polynomial time. By Corollary 10, deciding whether* $\{\varnothing, S\}$ *is a global attractor is* **coNP***-hard, thus the thesis follows.*

References

1. Ehrenfeucht, A., Rozenberg, G.: Basic notions of reaction systems. In: Developments in Language Theory, 8th International Conference (DLT). Volume 3340 of Lecture Notes in Computer Science., Springer (2004) 27–29
2. Formenti, E., Manzoni, L., Porreca, A.E.: On the complexity of occurrence and convergence problems in reaction systems. Nat. Comput. **14**(1) (2015) 185–191
3. Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E.: Ancestors, descendants, and gardens of eden in reaction systems. Theor. Comput. Sci. **608** (2015) 16–26
4. Azimi, S., Gratie, C., Ivanov, S., Manzoni, L., Petre, I., Porreca, A.E.: Complexity of model checking for reaction systems. Theor. Comput. Sci. **623** (2016) 103–113
5. Barbuti, R., Gori, R., Levi, F., Milazzo, P.: Investigating dynamic causalities in reaction systems. Theor. Comput. Sci. **623** (2016) 114–145
6. Nobile, M.S., Porreca, A.E., Spolaor, S., Manzoni, L., Cazzaniga, P., Mauri, G., Besozzi, D.: Efficient simulation of reaction systems on graphics processing units. Fundam. Informaticae **154**(1-4) (2017) 307–321
7. Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E.: Complexity of the dynamics of reaction systems. Inf. Comput. **267** (2019) 96–109
8. Barbuti, R., Bernasconi, A., Gori, R., Milazzo, P.: Characterization and computation of ancestors in reaction systems. Soft Comput. **25**(3) (2021) 1683–1698
9. Okubo, F., Kobayashi, S., Yokomori, T.: Reaction automata. Theoretical Computer Science **429** (2012) 247–257 Magic in Science.
10. Okubo, F., Yokomori, T. In: The Computing Power of Determinism and Reversibility in Chemical Reaction Automata. Springer International Publishing, Cham (2018) 279–298
11. Yokomori, T., Okubo, F.: Theory of reaction automata: a survey. J. Membr. Comput. **3**(1) (2021) 63–85
12. Okubo, F., Fujioka, K., Yokomori, T.: Chemical reaction regular grammars. New Gener. Comput. **40**(2) (2022) 659–680

13. Brodo, L., Bruni, R., Falaschi, M., Gori, R., Levi, F., Milazzo, P.: Quantitative extensions of reaction systems based on SOS semantics. Neural Comput. Appl. **35**(9) (2023) 6335–6359
14. Corolli, L., Maj, C., Marini, F., Besozzi, D., Mauri, G.: An excursion in reaction systems: From computer science to biology. Theor. Comput. Sci. **454** (2012) 95–108
15. Azimi, S., Iancu, B., Petre, I.: Reaction system models for the heat shock response. Fundam. Informaticae **131**(3-4) (2014) 299–312
16. Ivanov, S., Petre, I.: Controllability of reaction systems. J. Membr. Comput. **2**(4) (2020) 290–302
17. Barbuti, R., Bove, P., Gori, R., Gruska, D.P., Levi, F., Milazzo, P.: Encoding threshold boolean networks into reaction systems for the analysis of gene regulatory networks. Fundam. Informaticae **179**(2) (2021) 205–225
18. Barbuti, R., Gori, R., Milazzo, P.: Encoding boolean networks into reaction systems for investigating causal dependencies in gene regulation. Theor. Comput. Sci. **881** (2021) 3–24
19. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Functions defined by reaction systems. Int. J. Found. Comput. Sci. **22**(1) (2011) 167–178
20. Manzoni, L., Pocas, D., Porreca, A.E.: Simple reaction systems and their classification. International Journal of Foundations of Computer Science **25**(04) (2014) 441–457
21. Ascone, R., Bernardini, G., Manzoni, L.: Fixed points and attractors of additive reaction systems. Natural Computing (2024) 1–11
22. Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E.: Reachability in resource-bounded reaction systems. In: Language and Automata Theory and Applications: 10th International Conference (LATA), Springer (2016) 592–602
23. Azimi, S.: Steady states of constrained reaction systems. Theor. Comput. Sci. **701** (2017) 20–26
24. Ascone, R., Bernardini, G., Manzoni, L.: Fixed points and attractors of reactantless and inhibitorless reaction systems. Theoretical Computer Science **984** (2024) 114322
25. Formenti, E., Manzoni, L., Porreca, A.E.: Cycles and global attractors of reaction systems. In: Descriptional Complexity of Formal Systems: 16th International Workshop (DCFS), Springer (2014) 114–125
26. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction systems with duration. Computation, cooperation, and life **6610** (2011) 191–202
27. Granas, A., Dugundji, J.: Elementary fixed point theorems. In: Fixed Point Theory. Springer New York, New York (2003) 9–84
28. Papadimitriou, C.: Computational Complexity. Theoretical computer science. Addison-Wesley (1994)

Synchronization of rules in membrane computing

Péter Battyányi

Department of Computer Science,
Faculty of Informatics, University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
e-mail: battyanyi.peter@inf.unideb.hu

Summary. In this paper, computational models that are variants of membrane systems with synchronization of rules in the style of Aman and Ciobanu [1] are considered. We examine membrane system like computational models in different execution modes and with reusage and without reusage of objects in the same computational step. The weak cases are the ones without restrictions on the compound rules. We show in one of the cases that, as a computational model, it is strictly weaker than Turing machines when maximally parallel execution mode is omitted. Furthermore, we prove that the strong cases, when additional conditions are imposed on the compound rules, are computationally complete even without maximal parallelism. Finally, we give a more or less intuitive argument on why these computational systems with non-cooperative rules cannot be computationally complete even in the strong modes. **Keywords:** Membrane systems, Computational completeness, Rewriting systems

1 Introduction

Membrane systems, or P systems, are biologically inspired models of computation introduced by Gh. Păun in [9]. The original model is based on a tree-like structure of nested membranes. The computation proceeds separately in each region: the membranes or regions have their associated multisets that evolve in accordance with various rewriting rules specific to each membrane. In most of the cases, the whole process is synchronized by a global clock: each membrane is waiting for the other one to finish their computation before a new computational step begins. Originally, the computation in each membrane follows a maximally parallel mode, which means that a maximal multiset of rules is applied at the same time, that is, in each membrane, a multiset of rules is executed simultaneously which is such that no more rules could have been added to the multiset to maintain the simultaneous execution property. Several variants of P systems and application modes have been introduced and studied, we refer the interested reader to the monograph [10] for a thorough introduction, or the handbook [11] for a summary of notions and results of the area.

In this paper, we consider variants of the symbol object P system with several types of execution mode and forms of rules. Namely, besides the rules of the form $u \to v$, where u is the multiset to be replaced by the multiset v during a rule application, we consider synchronized rules in the sense introduced by Aman and Ciobanu [1, 2]. We pose the question of what is the computational power of these rules when we consider them with various execution modes and semantics. One of the execution modes is the unsynchronized or sequential one, where the rules can be applied one after the other. The other one is the synchronized mode where a (possibly empty) multiset of rules is executed simultaneously at the same time in a specific membrane computational step in each compartment. We remove, however, the requirement for the rule applications of being maximally parallel. Observe that the unsynchronized mode can be considered as rule executions when the newly obtained objects can be reused in the next computational step, while the synchronized mode is such execution of rules where the newly obtained objects coming from the right hand sides of the rules can only be reused in the next computational step. Regarding the semantics, we even distinguish two different interpretations concerning the synchronized rules. We term them weak and strong application modes. In total, we will talk about four different execution modes: unsynchronized and synchronized modes with the weak or with the strong application mode. It will turn out that the weak application modes are strictly weaker than the strong ones, the latter ones being equivalent to the computational power of the Turing machine model.

2 Preliminaries

2.1 Multisets

Let $\mathbb{N}$ and $\mathbb{N}_{>0}$ be the set of non-negative integers and the set of positive integers, respectively, and let O be a finite nonempty set (the set of object). A *multiset* M over O is a pair $M = (O, f)$, where $f : O \to \mathbb{N}$ is a mapping which gives the *multiplicity* of each object $a \in O$. If $f(a) = 0$ for every $a \in O$, then M is the empty multiset. If $f(a) = n > 0$, then $a \in M$, or $a \in^n M$.

Let $M_1 = (O, f_1), M_2 = (O, f_2)$. Then $(M_1 \sqcap M_2) = (O, f)$ where $f(a) = \min\{f_1(a), f_2(a)\}$; $(M_1 \sqcup M_2) = (O, f')$, where $f'(a) = \max\{f_1(a), f_2(a)\}$; $(M_1 \oplus M_2) = (O, f'')$, where $f''(a) = f_1(a) + f_2(a)$; $(M_1 \ominus M_2) = (O, f''')$ where $f'''(a) = \max\{f_1(a) - f_2(a), 0\}$; and $M_1 \sqsubseteq M_2$, if $f_1(a) \leq f_2(a)$ for all $a \in O$. We abbreviate $\underbrace{M \oplus M \oplus \ldots \oplus M}_{k}$ as $k \cdot M$.

The number of copies of objects in a finite multiset $M = (O, f)$ is its cardinality: $\mathrm{card}(M) = \Sigma_{\{a|f(a)>0\}} f(a)$. Such an M can be represented by any string w over O for which $|w| = \mathrm{card}(M)$, and $|w|_a = f(a)$ where $|w|$ denotes the length of the string, and $|w|_a$, or simply $w(a)$, denotes the number of occurrences of the symbol a in w.

We define the $\mathcal{MS}^n(O)$, $n \in \mathbb{N}$, to be the set of all multisets $M = (O, f)$ over O such that $f(a) \leq n$ for all $a \in O$, and we let $\mathcal{MS}^{<\infty}(O) = \bigcup_{n \geq 0} \mathcal{MS}^n(O)$. Moreover, if A is an arbitrary set, we define $A^{\geq k} = \cup_{n=k}^{\infty} A^n$, where $A^0 = \emptyset$, $A^1 = A$ and $A^n = \underbrace{A \times \ldots \times A}_{n}$ for $n \geq 2$.

If O is a set of objects and $u, v \in \mathcal{MS}^{<\infty}(O)$, we call the (u, v) a rule over O. In what follows, we write $\mathcal{MS}(O)$ in place of $\mathcal{MS}^{<\infty}(O)$ since we will exclusively deal with finite multisets.

2.2 Symbol object P systems of degree 1

Since the concepts that we will study in the sequel are in compliance with the construction of flattening of membrane systems [3], in the sequel, we restrict our attention to membrane systems of degree 1. Below, we provide the definition of a symbol object P system of degree 1.

A *P system* of degree 1 is a tuple $\Pi = (O, w_0, R)$ where O is an alphabet of objects, $w_0 \in MS(O)$ is the initial content of the region, R is the set of (evolution) rules associated with region 1. They are of the form $u \to v$, where $u, v \in \mathcal{MS}(O)$. We assume that the result of the computation is collected from membrane 1 in the form of a multiset of certain terminal objects. A computation gives a result when it comes to a halt. For a rule $r = u \to v \in R_i$, we write $lhs(r)$ for u and $rhs(r)$ for v. A configuration w is the actual multiset content of membrane 1.

2.3 P systems with synchronized rules

In this subsection we present the various versions of P systems with synchronized rules. A P system with with rule synchronization, or P system with synchronized rules, (of degree 1) is a tuple $\Pi = (O, w_0, (R, \rho))$, where $\Pi = (O, w_0, R)$ is a P system of degree 1 and $\rho \subseteq R^{\geq 2}$. For any element $(r_1, r_2, \ldots, r_k)$ of ρ, where $r_i \in R$ $(1 \leq i \leq k)$, we use the notation $r = r_1 \otimes r_2 \otimes \ldots \otimes r_k$ and we call r a synchronized rule, or a compound rule, and r_i its components $(1 \leq i \leq k)$. Let $comp(r)$ denote the set of components of r. We term $r \in R$ a single rule if it is not a synchronized one. In the sequel, we consider P systems with rule synchronization of degree 1. We say that a rule $r \in R$ is non-cooperative if it is of the form $a \to v$ for some $a \in O$. Otherwise, r is said to be cooperative. Similarly, a rule $\rho = r_1 \otimes \ldots \otimes r_k$ is non-cooperative if each of $r_1, \ldots, r_k$ is of the form $a \to v$ for some object $a \in O$.

We clarify how rule execution can be understood in P systems $\Pi = (O, w_0, (R, \rho))$ with rule synchronization. We distinguish two kinds of rule application modes-weak and strong application modes-, and two possibilities for dealing with objects appearing on the right hand side of rules: we can either allow reusing objects created by a rule application or prohibit this in the same computational step. The latter corresponds to the original interpretation in membrane computation. In total, we deal with four possible ways to interpret P systems with synchronized rules. Let w be a configuration of Π. In what follows, we describe how the

next configuration w' emerges from w. Let $r \in R$ be a single rule or let r be a component of a compound rule r'. Then we say that r is applicable in w if $lhs(r) \sqsubseteq w$. An application of a single rule involves the replacement of w with the multiset $w' = w \ominus lhs(r) \oplus rhs(r)$. The applicability and the result of application of compound rules will be clarified in the respective execution modes.

Definition 1.

(W1) *(Weak application mode with reusage of objects.) Reusage of objects means there is no synchronization in the P system, the rules are applied in a sequential manner. Let us clarify what application of a rule means in the specific cases. Let $r = u \to v \in R$. Then r is applicable if $u \sqsubseteq w$. In this case, the result of applying r to w is obtained by removing form w the objects of u and adding the objects of v to $w \ominus u$. Now, let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_n \in \rho$, where $r_i = u_i \to v_i$ $(1 \le i \le n)$. Then r is applicable iff all of its components are applicable as single rules. When this is the case, an application of r consists of applications of the components of r an arbitrary number of times provided all of them are applied at least once. Moreover, an object emerging on the right hand side of a component can be reused even if the execution step of r has not finished yet. More formally, let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_n \in \rho$. Then r is applicable if $u_1 \oplus \ldots \oplus u_n \sqsubseteq w$. In this case, the result of the application, w', is obtained through a sequence of intermediate configurations $w_1, w_2, \ldots, w_k$, where $w = w_1 \to^{r'_1} w_2 \to^{r'_2} \ldots \to^{r'_{k-1}} w_k = w'$ and $w_{j+1} = w_j \ominus lhs(r'_j) \oplus rhs(r'_j)$ $(1 \le j \le k)$ and each of $r_1, \ldots, r_n$ occurs in the sequence $r'_1, \ldots, r'_k$ at least once. We call the configurations w and w' proper and the configurations $w_2, \ldots, w_{k-1}$ intermediate ones. The transitions $w_j \to^{r'_j} w_{j+1}$ are small step transitions $(1 \le j \le k-1)$, while the transition yielding w' from w is a big step transition. In notation: $w \Rightarrow^r_{w,y} w'$.*

(W2) *(Weak application mode without reusage of objects.) In this case, a big step comprises the simultaneous application of several single or compound rules in the compartments. When all the membranes has finished working, only then can the next computational step begin. The objects coming from the right hand side of the rules can be used only in the next big computational step. Formally, let $\mathcal{R}$ be a multiset over the set of rules $R \cup \rho$. We define multisets, $sub(\mathcal{R})$ and $add(\mathcal{R})$ over O. Firstly, let $r = u \to v \in R$. Then $sub(r) = lhs(r) = u$ and $add(r) = rhs(r) = v$. On the other hand, if $r = r_1 \otimes \ldots \otimes r_n \in \rho$, let $r_i = u_i \to v_i$ $(1 \le i \le n)$. Then $sub(r) = \bigoplus_{i=1}^n k_i \cdot u_i$ and $add(r) = \bigoplus_{i=1}^n k_i \cdot v_i$, where $k_i \ge 1$ $(1 \le i \le n)$. We write $sub(\mathcal{R}) = \bigoplus\{sub(r) \mid r \in \mathcal{R}\}$ and $add(\mathcal{R}) = \bigoplus\{add(r) \mid r \in \mathcal{R}\}$ and we set $w' = (w \ominus sub(\mathcal{R})) \oplus add(\mathcal{R})$.*

(S1) *(Strong application mode with reusage of objects.) This application mode demands that the application of rules, being either single or compound ones, takes place in a sequential way. The only difference regarding application mode (W1) is the different interpretation of the compound rules. Let $r \in R$ be a single rule. Then r is applicable if $lhs(r) \sqsubseteq w$ and, in this case, $w' = w \ominus lhs(r) \oplus rhs(r)$. On the other hand, let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_n \in \rho$. Then r is applicable to w_0 in the strong sense if at least one of r's components is applicable. When this holds,*

we apply r by searching from left to right the first applicable component r'. After executing r', we resume searching for an applicable component of r starting from the first component. More formally, r is applicable if $lhs(r_i) \sqsubseteq w_0$ for some $1 \leq i \leq n$. Then an application of r is described by the following sequence of intermediate configurations: $w_0 \to^{r'_1} w_1 \to^{r'_2} w_2 \ldots \to^{r'_k} w_k = w'$, where r'_i is of minimal index i among the components of r applicable to w_{i-1} $(1 \leq i \leq k)$. No component is applicable to $w' = w_k$. The transitions $w_{i-1} \to^{r'_i} w_i$ $(1 \leq i \leq k)$ are called small steps regarding the application of r in the strong sense with reusage of objects, while the process yielding w' form w is called a big step. In notation: $w \Rightarrow^r_{s,y} w'$. The configurations w and w' are proper.

(S2) *Strong application mode without reusage of objects.) It differs the above mode only in the treatment with the objects coming form the right hand side of the components during an execution of a compound rule. In plain words, a compound rule r is executed by searching for the first applicable component with the smallest index. Then the component is executed as a single rule, i.e., we subtract the multiset on the left hand side from the actual configuration and add the multiset on the right hand to the result of the subtraction. Then we start searching for the applicable component with the smallest index in the emerging new multiset. This process stops until there are no more applicable components. More precisely, let $r \in R$. Then let the result of the transition $w \Rightarrow^r_{s,n} w'$ be the multiset $w' = w \ominus lhs(r) \oplus rhs(r)$. Suppose $r \in \rho$. Then we define the multisets $lhs(r)$, $rhs(r)$ by giving three sequences of multisets $u_0, u_1, \ldots$, $v_0, v_1, \ldots$ and $w_0, w_1, \ldots$ simultaneously. Let $u_0 = \varepsilon$, the empty multiset, and let $v_0 = w_0 = w$. Assume u_i, w_i, v_i are already defined for some $i \in \mathbb{N}$. Let r' be the component of r that is the first one from left to right which is applicable to w_i. Then $u_{i+1} = u_i \oplus lhs(r')$ and $w_{i+1} = w \ominus u_{i+1}$, $v_{i+1} = v_i \oplus rhs(r')$. We continue if at least one component of r is applicable, that is, we calculate the $i + 2$-th elements of the sequences if $lhs(r'') \sqsubseteq w_{i+1}$ for some $r'' \in comp(r)$. Let m be the first index for which this is not case. Then we denote $lhs(r) = u_m$ and $rhs(r) = v_m$. We have $w' = w \ominus lhs(r) \oplus rhs(r)$, and we write $w \Rightarrow^r_{s,n} w'$.*

Let us illustrate the definition by demonstrating the operation of a P system with synchronization of rules in the strong application mode with reusage of objects (S2).

Example 1. Let $\Pi = (O, w_0, (R, \rho))$ be a P system with synchronization of rules, where $n = 1$, $O = \{a, b, d, e\}$, $w_0 = a^n b^m$, and $R = \{r_1 = ad^m \to e^m b^m, r_2 = ab \to da, r_3 = b \to \varepsilon\}$. Let $\rho = \{r = r_1 \otimes r_2 \otimes r_3\}$. Let us consider a terminating computation starting from $w_0 = a^n b^m$ in mode (S2). We assume $n, m \geq 1$. Let the subscripts of the arrows denote the components of r applied.

$$
\begin{aligned}
&a^n b^m \to^*_{r_2} a^n d^m \to_{r_1} a^{n-1} e^m b^m \to_{r_2} \\
&a^{n-1} e^m b^{m-1} d \to^*_{r_2} a^{n-1} e^m d^m \to_{r_1} a^{n-1} e^{2m} b^m \to_{r_2} \\
&\ldots \\
&a e^{(n-1)m} b^{m-1} d \to^*_{r_2} a e^{(n-1)m} d^m \to_{r_1} e^{nm} b^m \to^*_{r_3} e^{nm}
\end{aligned}
$$

In what follows, let w stand for the actual configuration of our membrane. In the example above, when the multiset w contains less than m d's, then the successive applications of the component r_2 remove one copy of b and add a d to w. The new copies of d add to the configuration. In the case when $d^m \sqsubseteq w$, r_1 is applicable and, hence, it must be applied. This means an erasure of one copy of a and introducing e^m and b^m to w. The computation proceeds in this way until all the a's are consumed. At this point, $w = e^{nm}b^m$. Now only r_3 can be applied, which yields the removal of b^m from w.

3 The power of synchronized rules in P systems

We turn to a brief discussion of the computational power of synchronization of rules in P systems with respect to the above application modes. We treat first the case of weak application modes. In this section, we present some results and provide their short justifications or we give hints on how they can be achieved.

3.1 Synchronized rules with the weak application mode

In this subsection we examine P systems with synchronization of rules using the weak application mode. It turns out that P systems with application mode (W1) are not computationally complete, and we formulate a conjecture for a similar statement on P systems with application mode (W2). In the case of application mode (W1), that is, the weak application mode with reusage of objects, we show that compound rules can be substituted for ordinary rules of a P system such that the multisets computed by the two P systems will be the same.

Proposition 1. *Let $\Pi = (O, w_0, (R, \rho))$ be a P system of degree 1 with synchronization of rules using execution mode (W1). Then there exists a P system Π' of degree 1 without synchronization of rules applying the sequential mode such that Π' and Π compute the same sets of vectors.*

Proof. Let $\Pi = (O, w_0, (R, \rho))$ be as above. Let us define Π' in the following way. Let us add to O a finite set of new objects as described below:

$$O' = O \cup \{\omega, \kappa, \vartheta_1^r, \ldots, \vartheta_{k_r}^r \mid r = r_1 \otimes \ldots \otimes r_{k_r} \in \rho\}.$$

Let $w_0' = w_0 \sqcup \{\omega, \vartheta_1^r \mid r \in \rho\}$. We obtain R' as follows. Firstly, we add the following single rules to R for any $r = r_1 \otimes \ldots \otimes r_k \in \rho$. Let $r_i = u_i \to v_i$ $(1 \leq i \leq k)$. We define the rules:

$$
\begin{aligned}
r'_1 &= \omega u_1 \vartheta_1^r \to \kappa v_1 \vartheta_1^r,\\
r''_1 &= \kappa u_1 \vartheta_1^r \to \kappa v_1 \vartheta_2^r,\\
r'_2 &= \kappa u_2 \vartheta_2^r \to \kappa v_2 \vartheta_2^r,\\
r''_2 &= \kappa u_2 \vartheta_2^r \to \kappa v_2 \vartheta_3^r,\\
&\dots\\
r'_k &= \kappa u_k \vartheta_k^r \to \kappa v_k \vartheta_k^r,\\
r''_k &= \kappa u_k \vartheta_k^r \to \omega v_k \vartheta_1^r.
\end{aligned}
$$

We add the rule $\kappa \to \kappa$ to R'. Moreover, if $r = u \to v \in R$, then we let $r' = \omega u \to \omega v \in R'$. It is easy to check that $\Pi' = (O', w'_0, R')$ produces the same set of multisets as Π. □

Regarding the case of (W2), i.e., weak application mode without reusage of objects we believe that the computational power is strictly weaker than that of Turing machines. We assert this as a conjecture. We think that a P system with application mode (W2) can be simulated with a forbidding context grammar with λ-rules.

Conjecture 1. Let $\Pi = (O, w_0, (R, \rho))$ be a P system of degree 1 with synchronization of rules applying execution mode (W2). Then there exists a forbidding context grammar G_Π with λ-rules such that $Ps(\Pi) = Ps(G_\Pi)$.

3.2 Synchronized rules with the strong application mode

Now we continue with our investigation with synchronization of rules in the strong application mode. We demonstrate that generalized P systems with synchronized rules in the strong application mode can compute any Turing computable, or in, other words, partial recursive function. To this end, we simulate register machines with zero-test subtraction with P systems applied with the strong application mode. We recall briefly the definition of such register machines.

A *register machine* is a tuple $W = (m, H, l_0, l_h, \text{Inst})$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and Inst is the set of instructions. There is a bijection between the labels of H and the instruction of Inst. The following types of instructions can be used. For $l_i, l_j, l_k \in H$ and $r \in \{1, \dots, m\}$ we have:

- l_i : (ADD(r), l_j, l_k) - *nondeterministic add*: Add 1 to register r and then go to one of the instructions with labels l_j or l_k, nondeterministically chosen.
- l_i : (SUB(r), l_j, l_k) - *zero check and subtract*: If register r is empty, then go to the instruction with label l_j, if r is non-empty, then subtract one from it and go to the instruction with label l_k.
- l_h : HALT - *halt*: Stop the machine.

A computation of a register machine starts with all registers empty except for some designated input registers. The control flow is regulated by the labels shown

in the actual instruction of the machine. The machine starts with the instruction labeled l_0. If the machine reaches the halt instruction l_h : `HALT`, then it stops, and the number stored in the first register, or in the output registers fixed in advance, is the result of the computation. Note that our register machine is a nondeterministic computing device. In this case, it is suitable for computing sets of natural numbers when we consider the outcomes of the various computations.

We consider the two computation modes, (S1) and (S2), separately. Firstly, we deal with (S1), that is, strong computation mode with reusage of objects.

Theorem 1. *Generalized P systems with synchronized rules in the strong application mode with reusage of objects can simulate arbitrary register machines with zero-test subtractions, even without using the maximally parallel rule execution. We may even assume that the P system is a purely catalyctic one with a three-state catalyst.*

Proof. Let $W = (m, H, l_0, l_h, \mathrm{Inst})$ be a register machine. Fpr the sake of simplicity, we assume that W has one output register, let this be register 1, W computes a number instead of a vector, and, in addition, W starts its computation with all registers initially empty. We construct a P system Π of degree 1 with synchronization of rules using execution mode (S1) such that, at every computational step, the number stored in register i of W is represented by the number of the object a_i in the only region of Π and, upon halting, Π provides the result by producing the same number of copies of object a_1 as the number stored in register R_1 of W.

We define Π as a purely catalyctic P system with a three-state catalyst. We have to take care that the execution mode allows us reusing elements created during a rule application, in contrast with common practice in membrane systems. Let $\Pi = (O, w_0, (R, \rho))$, where

$$\begin{aligned} O &= \{l \mid l \in H\} \cup \{a_r \mid 1 \le r \le m\} \cup \{c, c', c''\}, \\ w_0 &= l_0 c, \\ R &= R_{Add} \cup R_{Sub} \cup R_{Halt}, \end{aligned}$$

where

$$R_{Add} = \{r_{(i,1)} = cl_i \to cl_j a_r, r_{(i,2)} = cl_i \to cl_k a_r,$$
$$r''_{(i,1)} = c''l_i \to c''l_j a_r, r''_{(i,2)} = c''l_i \to c''l_k a_r \mid \text{for all}$$
$$l_i : (\texttt{ADD}(r), l_j, l_k) \in \text{Inst}\},$$

$$R_{Sub} = \{r^o_{(i,1)} = ca_r \to c', r^o_{(i,2)} = c'l_i \to c''l_j, r^o_{(i,3)} = cl_i \to c''l_k,$$
$$r^e_{(i,1)} = c''a_r \to c', r^e_{(i,2)} = c'l_i \to cl_j, r^e_{(i,3)} = c''l_i \to cl_k$$
$$\mid \text{for all } l_i : (\texttt{SUB}(r), l_j, l_k) \in \text{Inst}\},$$

$$R_{Halt} = \{cl_h \to c, c''l_h \to c''\},$$

$$\rho = \{\rho^o_i = r^o_{(i,1)} \otimes r^o_{(i,2)} \otimes r^o_{(i,3)}, \rho^e_i = r^e_{(i,1)} \otimes r^e_{(i,2)} \otimes r^e_{(i,3)} \mid l_i : (\texttt{SUB}(r), l_j, l_k)\}.$$

We give a brief explanation of how Π simulates a computation of W. The initial configuration corresponds to the initial configuration of W, since the first region contains l_0, the label of the starting instruction, and W commences its operation with the assumption that all registers are initially empty. The label of the next instruction, together with copies of the objects a_r ($1 \leq r \leq m$), are to be found in membrane 1, the only membrane of Π. There are two sets of rules with respect to the SUB instruction: they correspond to the odd and even turns of the applications of SUB. When in the instruction sequence the SUB has been called an even number of times, the next execution will be governed by a ρ-rule with superscript "o". Otherwise, it is the turn of the ρ-rules with superscript "e". This will be detailed below. We describe the next step of the simulation process by taking into account the different cases. Let $conf$ denote the actual configuration of Π, that is, the content of membrane 1.

- $l_i : (\texttt{ADD}(r), l_j, l_k)$. The applications of the corresponding rules of Π introduce the label representing the next instruction of W in Π, adding one copy of a_r to $conf$ at the same time. We distinguish the different cases regarding the actual number of SUB instructions applied before calling instruction l_i.
- $l_i : (\texttt{SUB}(r), l_j, l_k)$. Assume an even number of SUB has already been applied and it is the turn of the instruction l_i. Then $c \in conf$, and the rule ρ^o_i is executed. If $a_r \in conf$, then the object c' is introduced and, in the next step, l_i is transformed to l_k and c' is exchanged with c''. At this point, the operation of ρ^o_i halts and the simulation of the instruction labelled l_k can commence. If $a_r \notin conf$, then $r^o_{(i,1)}$ cannot be applied, instead, a copy of c'' and l_k is introduced using $r^o_{(i,3)}$. If l_i emerges as an even turn of an application of SUBs, then $c'' \in conf$ and a process similar to the above one can be executed.
- $l_h : \texttt{HALT}$. Then l_h is removed and the computation of Π comes to a halt.

Obviously, the P system Π halts if and only if the register machine W halts and, at this point, the number of copies of the object a_1 and the number stored in the first register of W are the same. □

We formulate a similar statement asserting the possibility of simulation register machines with P systems in application mode (S2). In this case, it is enough to consider a purely catalyctic P system with bistable catalyst for the simulation. Since the proof is very much like in the case of application mode (S1), we just state the result without proof.

Theorem 2. *Generalized P systems with synchronized rules in the strong application mode without reusage of objects (S2) can simulate, without using the maximally parallel rule execution, arbitrary register machines with zero-test subtractions, even when we consider purely catalyctic P systems with a bistable catalyst.*

Proof. Similar to that of the previous theorem. This time we do not need to consider alternating turns in the application of instruction SUB, hence, the proof even simplifies a little. □

The above proofs demonstrate the fact that P systems with synchronization of rules and the strong application mode are able to simulate register machines either with reusage or without reusage of objects. The question naturally arises how far can we go in the simplification of rules constituting the P system. In what follows, we assert a claim saying that non-cooperative rules are not enough for ensuring computational completeness. We omit the more or less intuitive argument for the claim, however, it already provides us with a string justification that the statement holds. We formulate our assertion for the case (S1) only, the case for (S2) being similar, mutatis mutandis.

Theorem 3. *Let Π be a P system of degree 1 with synchronization of rules, with non-cooperative rules of execution mode (S1). Then Π is not computationally complete.*

Proof. [Sketch] The intuitive argument relies on the fact that Π cannot compute the $\overline{sg}$ function, where

$$\overline{sg} = \begin{cases} 1 \text{ if } n = 0, \\ 0 \text{ otherwise.} \end{cases}$$

More precisely, we show that, if $O = \{a_1, \ldots, a_n\}$ is the object set and $c = (c(a_1), \ldots, c(a_n))$ is a corresponding configuration of Π, when we are given two configurations c' and c'' such that $c' \sqsubseteq c''$ as multisets, then Π cannot evolve on c' and c'' such that, upon halting, the reverse relation would hold. I.e., when $c'' \Rightarrow^* \underline{c}''$ and $\underline{c}''$ is a hlating configuration then there exists a halting configuration $\underline{c}'$ such that $c' \Rightarrow^* \underline{c}'$. Since Π computes a function, it cannot be the case that the result for the input c' is represented by a configuration c for which $c \sqsubseteq \underline{c}''$ does not hold. □

4 Concluding remarks

- The problems discussed in the paper stem from a specific membrane system defined by Aman et al. [1]. Our results partly deviate from the usual membrane

system notions by examining rule applications with the possibility of reusage of elements even in the same computational step. The execution modes (W1) and (S1) could have equally been formulated for computational models with this property, e.g., for Petri nets.

- In some programming languages, like SML [4], lines are processed from top to bottom. This imports lends some control facilities to program execution. The present results are in accordance with this experience of programmers: the weak application mode could be associated with a purely declarative philosophy of program execution, while the strong application mode can be related to an implementation where the order of instructions matter. Our result intimates that setting up an order of execution for the rules adds computational strength to the programming language implementation.
- The P systems constructed for the simulation of register machines look very much like generalized communicating P systems (GCPS) in appearance [5]. GCPSs posses a graph-like structure, where each node, called a cell, contains a multiset of objects which may move between the cells by the so-called communication rules. A communication rule has the form $(a, i)(b, j) \to (c, k)(d, l)$, where a, b, c, d are objects and i, j, k, l represent the input and output regions, respectively. Depending on the values of the identifiers i, j, k, l, several restricted forms of interaction rules can be specified. Generalized communicating P systems mostly obey the maximally parallel rule execution mode. It can be shown that, in most of the cases, GCPSs are Turing complete even with a set of restricted form of interaction rule and taking a relatively small, fixed number of cells [5]. Computational completeness is preserved when we consider an alphabet with one object and rules of restricted types or only as many as three cells together with rules of restricted types [6, 7]. It would be interesting to explore the similarities and differences between GCPSs and the computational model defined in this paper and obtain results of an analogous nature by restricting the form of the rules, the number of cells or the number of objects. We would emphasize the main difference between the two computational models: with synchronization of rules Turing completeness is achieved without additional control facilities in the strong application mode, namely, without imposing the necessity of maximally parallel execution mode. Hence, results of somehow different types should be expected in our case.

References

1. Aman, B., Ciobanu, G., Synchronization of rules in membrane computing. Journal of Membrane Computing **1**, 233–240 (2019). https://doi.org/10.1007/s41965-019-00022-1
2. Aman, B., Ciobanu, G., The power of synchronizing rules in membrane computing, Information Sciences **594**, 360–370 (2022)
3. Agrigoroaiei, O., Ciobanu, G., Flattening the transition P systems with dissolution In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds)

Membrane Computing. CMC 2010. LNCS, vol 6501. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-18123-8_7
4. Blume, M.,: The SML/NJ Compilation and Library Manager. Lucent Technologies, Bell Labs (2002). https://www.smlnj.org/doc/CM/new.pdf
5. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. Theoretical Computer Science **412**(1-2), 124–135 (2011), https://doi.org/10.1016/j.tcs.2010.08.020.
6. Csuhaj-Varjú, E., Verlan, S.: Computationally Complete Generalized Communicating P Systems with Three Cells. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing. CMC 2017. LNCS, vol 10725. Springer, Cham. https://doi.org/10.1007/978-3-319-73359-3_8
7. Csuhaj-Varjú, E., Vaszil, G., Verlan, S.: On Generalized Communicating P Systems with One Symbol. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds) Membrane Computing. CMC 2010. Lecture Notes in Computer Science, vol 6501. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-18123-8_14
8. Alhazov, A., Belingheri, O., Freund, R., Ivanov, S., Porreca, A. E., and Zandron, C. (2016). Semilinear Sets, Register Machines, and Integer Vector Addition (P) Systems. In: Proceedings of 17th International Conference on Membrane Computing (CMC17), 39–56 http://hdl.handle.net/20.500.12708/56909
9. Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000)
10. Păun, G.: Membrane Computing: An Introduction. Springer-Verlag, Berlin, Heidelberg (2002)
11. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)

(Very) Initial Ideas on Non-cooperative Polymorphic P Systems and Parallel Communicating ET0L Systems

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. In this research, we begin to investigate the relationship between polymorphic P systems and parallel communicating ET0L systems. Our goal is to present an example, based on the definitions, from which it seems that there is some connection between the two systems.

Key words: Polymorphic P systems, P systems with non-cooperative rules, parallel communicating grammar system, parallel communicating ET0L systems

1 Introduction

Membrane systems (P systems) are computational models whose computation is based on the processes taking place in living cells. They consist of several nested membranes, these are called regions. The contents of the regions are multisets. In each step, we apply rule(s) in each region (if applicable), so we apply multiple rewriting rules in parallel until we reach a halting configuration.

The difference between polymorphic P systems and P systems is the relationship between multisets and rules. In polymorphic P systems, the contents of the regions form the rules. As the contents of the regions change, the corresponding rules also change, we call these dynamic rules. Each rule has two regions that make up the left and right sides of the rule. For more information, see the survey [1].

The results of the article [2] demonstrate the power of the model. in the case of using cooperative rules, any recursively enumerable set of numbers is generated. As a result, we deal with the non-cooperative case, which generates languages, interesting, mainly from the point of view that exponential, even super-exponential growth of the number of objects within the system can be achieved.

In this article, we begin to investigate the relationship between non-cooperative polymorphic P systems and parallel communicating ET0L systems. In the follow-

ing, after the necessary preliminaries and definitions in section 2, we present an example in section 3. Through this example, we show that it is possible that some kind of relationship exists between the two models, based on the definitions. We create a parallel communicating ET0L system that simulates the computation of a simple non-cooperative polymorphic P system.

2 Preliminaries and Definitions

In this section, we define the basic definitions and notions we will use. For more information about formal language theory, see [3], and [4, 5] for details about membrane computing.

First, we define the formal alphabet. An *alphabet* V is a finite non-empty set of symbols called letters. A string (or word) over V is a finite sequence of letters, the set of all strings over V is denoted by V^*, and $V^+ = V^* \setminus \{\lambda\}$ where λ denotes the empty string. If we fix an order $V = \{a_1, a_2, \ldots, a_n\}$ of the letters, then the vector $(|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$ is called the Parikh vector of the word $w \in V^*$.

If $\mathbb{N}$ denotes the set of nonnegative integers, then a *multiset* over a set U is a mapping $M : U \to \mathbb{N}$ where $M(a)$, for all $a \in U$, is the multiplicity of element a in the multiset M. If U is finite, $U = \{a_1, a_2, \ldots a_n\}$, then M 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) where a^j denotes the string obtained by concatenating $j \in \mathbb{N}$ occurrences of the letter $a \in V$ (with $a^0 = \lambda$).

A. Lindenmayer introduced Lindenmayer systems (L systems for short) in 1968. He introduced these systems with the aim of being able to describe the development of organisms known from biology using formal languages. L systems are parallel rewriting systems, see [6, 7] for more information on this area.

In the following, we define a version of the L systems, the ET0L systems, which are extended, tabled, and interactionless versions of the original L systems.

An *ET0L system* is a quadruple $G = (V, T, U, \omega)$ where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $\omega \in V^+$ is the initial word of G, and $U = (P_1, \ldots, P_m)$ where $P_i,\ 1 \leq i \leq m$, are finite sets of context-free productions over V (called *tables*), such that for each $a \in V$, there is at least one rule $a \to \alpha,\ \alpha \in V^*$ in each table.

In each computational step, G rewrites all the symbols of the current sentential form with the rules of one of the tables in U. The language generated by G consists of all terminal strings which can be generated in a series of computational steps (a derivation) starting from the initial word.

Let $L(G)$ be the language generated by G, then $L(G) = \{u \in T^* \mid w \Rightarrow^* u\}$ where $\Rightarrow$ denotes a computational step , and $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

We are not interested in the character string generated by the ET0L system as a sequence of letters, but only in the multiples of the different letters, i.e. the Parikh vectors of the words. This is necessary because we will connect the ET0L

languages to the multiset languages of the P systems. We denote by $Ps(G)$ the set of Parikh vectors corresponding the strings of $L(G)$ (Parikh set of $L(G)$), and by $PsET0L$ the class of Parikh sets corresponding to the class of languages generated by ET0L systems.

Polymorphic membrane systems were introduced in [2]. The rules in polymorphic P systems are defined by the contents of specific membrane regions corresponding to the left- and right-hand sides of the rule. As a result, the rules belonging to the regions change(s) during the computation. These rules are called dynamic rules.

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 of $2n+1$ membranes labelled by a symbol from the set $H = \{s, 1L, 1R, \ldots, nL, nR\}$, the elements of the multiset w_s are the initial contents of the skin membrane, the pairs of multisets $\langle w_{iL}, w_{iR} \rangle$ correspond to the initial contents of membranes iL and iR, $1 \leq i \leq n$, and $h_o \in H$ is the label of the output membrane.

The rules of the polymorphic membrane system are not given statically in the initial configuration. In each step, they are dynamically derived based on the contents of the left and right (iL and iR, $1 \leq i \leq n$) membrane pairs. Thus, if the membranes iL and iR belonging to the i-th membrane pair contain multisets u and v respectively, then in the next step we transform their parent membrane as if the multiset rewriting rule $u \to v$ were present.

If there is at least one rule in a system Π where the number of objects in u (the multiset on the left-hand side) can grow to be greater than one, then we say that Π is a *cooperative* system, otherwise, it is a *non-cooperative* system. The P system is a series of computational steps in which the rules belonging to the given region are applied in a maximal parallel way. Each object can be rewritten by at most one rule in one step. The P system reaches a halting configuration when no rule can be applied in any of the regions, so no more computational steps are possible.

The set of vectors $N(\Pi)$ generated by the polymorphic P system Π with the terminal alphabet T is the set of Parikh vectors among the strings $w \in T^*$ corresponding to the output h_o of multisets of terminal objects appearing in the region in a halting configuration Π, which is reached by computation starting in the initial configuration of the system.

We need the finitely representable (FIN-representable) property to define the possible objects belonging to the regions of the membrane system. FIN-representable property were introduced in [8]. Before describing the finitely representable property, we need a definition of σ^*.

Let $\Pi = (O, T, \mu, w_s, \langle w_{1L}, w_{1R} \rangle, \ldots, \langle w_{nL}, w_{nR} \rangle, h_o)$ be a polymorphic P system, and let $w_{j,h}$ denote the multiset after the jth step of the computation contained by the region labelled by h of Π for some $j \geq 0$, where $h \in \{s, 1L, 1R, \ldots, nL, nR\}$. We say that $w'_{j,h}$ is in the *successor set* of $w_{j,h}$, denoted

as $w'_{j,h} \in \sigma_{j,h}(w_{j,h})$, if $w'_{j,h}$ can be obtained from $w_{j,h}$ by the maximally parallel applications of the multiset rewriting rules associated to the region h.

If for the same $w_{j,h}$ as above, we fix $\sigma^0_{j,h}(w_{j,h}) = \{w_{j,h}\}$ for $k \geq 0$ and for any $j \geq 0$, we have $\sigma^{k+1}_{j,h}(w_{j,h}) = \sigma_{j+k,h}(\sigma^k_{j,h}(w_{j,h}))$, then we can define

$$\sigma^*_{j,h} = \bigcup_{k \geq 0} \sigma^k_{j,h}(w_{j,h}).$$

Given a polymorphic P system (Π), a region h of Π is *finitely representable* (or $FIN-representable$) if, starting from w_h, the multiset of initial objects of h, the set of successor multisets of w_h is finite, $\sigma^*_{0,h}(w_h)$ is finite.

We will need the definition of a finite transition system. Later on, we can represent the rules for the regions of the membrane system with state transitions. An finite transition system M can be denoted as a triple $M = (Q, q, \delta)$ where Q is a finite set of states, $q \in Q$ is the initial state, and $\delta : Q \to 2^Q$ is the state transition mapping. A state $q' \in \delta(q)$ is called the successor state of q, and $q \in Q$ is called a *halting state* if $\delta(q) = \emptyset$.

Parallel communicating grammar systems with Lindenmayer systems as components were introduced in [7]. In the following we recall the definition of parallel communicating ET0L systems (PC ET0L systems for short) based on [9].

A *parallel communicating grammar system* with n components is a (n+3) tuple

$$\Gamma = (N, K, T, G_1, \ldots, G_n), \text{ where}$$

N is a nonterminal alphabet, T is a terminal alphabet and K is an alphabet of query symbols ($K = \{Q_1, Q_2, \ldots, Q_n\}$). N, K and T are pairwise disjoint sets. In case the components are Lindenmayer systems, then $G_i = (N \cup K, T, P_i, \omega_i)$, where $1 \leq i \leq n$ with nonterminal and terminal alphabets as above, a table of rewriting rules in case of ET0L systems P_i, and an axiom $\omega_i \in (N \cup T)^*$. In most cases we call G_1 the master grammar of Γ.

The *language* generated by a parallel communicating system of extended Lindenmayer systems, $\Gamma = (N, K, T, G_1, \ldots, G_n)$, where $G_i = (N \cup K, T, P_i, \omega_i)$, $1 \leq i \leq n$, is

$$L(\Gamma) = \{\alpha_1 \in T^* | (\omega_1, \ldots, \omega_n) \Rightarrow^* (\alpha_1, \ldots, \alpha_n)\}$$

where G_1 is the master grammar of Γ.

3 Polymorphic P systems vs. PC ET0L systems

In this chapter, we start to examine the relationship between general non-cooperative polymorphic P systems and parallel communicating ET0L systems.

Using the definitions introduced in the previous chapter, we would like to show that probably the PC ET0L systems can generate the same class of languages as non-cooperative polymorphic P systems. Let's examine an example in which we construct a PC ET0L system for a non-cooperative polymorphic P system.

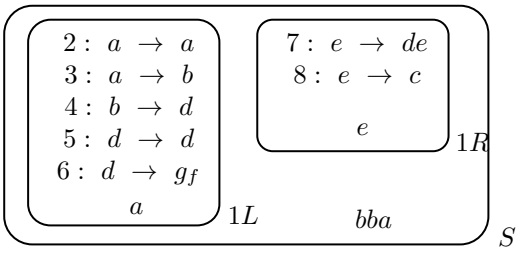


Fig. 1: The polymorphic P system Π of Example 1

Example 1. Consider a non-cooperative polymorphic P system

$$\Pi = (O, T, \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \ldots, \langle w_{8L}, w_{8R}\rangle, s)$$

where $O = T = \{a, b, c, d, e, g_f\}$ and the membrane structure is

$$\mu = [\ [\ldots]_{1L}\ [\ldots]_{1R}\ ]_s,$$

where the child membranes of 1L are $[\ ]_{2L}\ [\ ]_{2R} \ldots [\ ]_{6L}\ [\ ]_{6R}$, the children of 1R are $[\ ]_{7L}\ [\ ]_{7R}$, $[\ ]_{8L}\ [\ ]_{8R}$.

Let

$$w_s = bba,\ w_{1L} = a,\ w_{1R} = e,$$

and using the simplified notation for static (non-dynamic) rules, let the rules corresponding to 1L be

$$r_2 : a \to a,\ r_3 : a \to b,\ r_4 : b \to d,\ r_5 : d \to d, r_6 : d \to g_f,$$

and the rule corresponding to 1R be $r_7 : e \to de, r_8 : e \to c$, as illustrated in Figure 1.

According to the non-cooperative property, in each step 1-1 letter changes in $1L$. In the following we show that $1L$ is FIN-representable. Concerning $1L$, observe that $\sigma^*_{0,1L}(a) = \{a, b, d, g_f\}$ with $\sigma_{0,1L}(a) = \sigma_{j,1L}(a) = \{a, b\}$, $\sigma_{0,1L}(b) = \sigma_{j,1L}(b) = \{d\}$, $\sigma_{0,1L}(d) = \sigma_{j,1L}(d) = \{d, g_f\}$, and $\sigma_{0,1L}(g_f) = \sigma_{j,1L}(g_f) = \emptyset$ for all $j \geq 0$.

The $1R$ region is not FIN-representable, but from the point of view of the task, for us, the examination of the stopping of the regions on the right is not essential. It is sufficient that the FIN-representable property is true for the left sides, which will be true in all cases due to the non-cooperative property.

The skin region is not FIN-representable, as the dynamical rule r_1 given by the membranes labelled with $1L$ and $1R$ has more than one symbol on its right-hand side in each computational step, and this means that the number of symbols in the skin region is increasing with each rule application.

Following the steps introduced in [8], starting from the deepest (static) rules of the membrane system, we create the transition systems for the regions. To simplify

the example, we do not create transition systems for the static rules, but we know that they are the basis of the transition systems created for the higher levels. Due to the non-cooperativity and the FIN-representable property, a transition system can be constructed for each left-side membrane. Based on this, let's construct a transition system for the $1L$ region.

Start with the construction of $M_{2\ldots6} = (R_{2\ldots6}, \bar{r}_{2\ldots6}, \delta_{2\ldots6})$ as

- $R_{2\ldots6} = \{\bar{r}_{2\ldots6}\}$ where
- $\bar{r}_{2\ldots6} = (\bar{r}_2, \ldots, \bar{r}_6)$ with $\bar{r}_2 = ((a,\emptyset),(a,\emptyset))$, $\bar{r}_3 = ((a,\emptyset),(b,\emptyset))$, $\bar{r}_4 = ((b,\emptyset),(d,\emptyset))$, $\bar{r}_5 = ((d,\emptyset),(d,\emptyset))$, $\bar{r}_6 = ((d,\emptyset),(g_f,\emptyset))$, and
- $\delta_{2\ldots6}(\bar{r}_{2\ldots6}) = \emptyset$.

Note that the rule set corresponding to $\bar{r}_{2\ldots6}$ is $\{r_2, r_3, r_4, r_5, r_6\} = \{a \to a, a \to b, b \to d, d \to d, d \to g_f\}$.

Now we can construct M_{1L} transition system as follows: $M_{1L} = (Q_{1L}, q_0, \delta_{1L})$, where the set of possible states is $Q_{1L} = \{a, b, d, g_f\} \times \{(\bar{r}_2, \bar{r}_3, \bar{r}_4, \bar{r}_5, \bar{r}_6)\}$, that is,

$$Q_{1L} = \{q_0 : (a,\ (\bar{r}_2, \bar{r}_3, \bar{r}_4, \bar{r}_5, \bar{r}_6)),\ q_1 : (b,\ (\bar{r}_2, \bar{r}_3, \bar{r}_4, \bar{r}_5, \bar{r}_6)),$$
$$q_2 : (d,\ (\bar{r}_2, \bar{r}_3, \bar{r}_4, \bar{r}_5, \bar{r}_6)),\quad q_3 : (g_f,\ (\bar{r}_2, \bar{r}_3, \bar{r}_4, \bar{r}_5, \bar{r}_6))\},$$

the initial state is q_0, and the transition mapping is defined as

$$\delta_{1L}(q_0) = \{(q_0), (q_1)\},$$
$$\delta_{1L}(q_1) = \{(q_2)\},$$
$$\delta_{1L}(q_2) = \{(q_2), (q_3)\},$$
$$\delta_{1L}(q_3) = \emptyset.$$

It follows from this transition map that the state that starts with g_f is a final state. This shows that the $1L$ region is FIN-representable.

Construct a PC ET0L system that simulates the operation of the polymorphic membrane system. During construction, the current states of the FIN-representable regions must be recorded in PC ET0L sentential form.

Let us consider a PC ET0L system $\Gamma = (N, K, T, G_{s1L}, G_{1R}, G_m, G_c)$, simulating this membrane system.

In the following $|P_i|$ denote the number of tables in P_i, and $P_{i,j}$ denote the j-th table of P_i, $1 \le i \le n$, $1 \le j \le |P_i|$. $|P_{s1L}| = 4, |P_{1R}| = 1, |P_m| = 1, |P_c| = 1$. Let

$$N = \{q_0, q_1, q_2, q_3, F, S_{1R}, S_m, S_c\} \text{ non-terminals},$$
$$K = \{Q_{1R}, Q_{s1L}, Q_c\} \text{ query symbols and}$$
$$T = \{a, b, c, d, e, \} \text{ terminals}.$$

The initial string of the G_{s1L} component be ω_{s1L} and the associated tables are as follows:

$$\begin{aligned}
\omega_{s1L} &= bbaq_0,\\
P_{s1L,1} &= \{q_0 \to q_0, q_0 \to q_1, q_1 \to F, q_2 \to F, q_3 \to F, a \to Q_{1R}\},\\
P_{s1L,2} &= \{q_1 \to q_2, q_0 \to F, q_2 \to F, q_3 \to F, b \to Q_{1R}\},\\
P_{s1L,3} &= \{q_2 \to q_2, q_2 \to q_3, q_0 \to F, q_1 \to F, q_3 \to F, d \to Q_{1R}\}\\
P_{s1L,4} &= \{q_3 \to q_3, q_0 \to F, q_1 \to F, q_2 \to F\}
\end{aligned}$$

The initial string of the G_{1R} component be ω_{1R} and the associated tables are as follows:

$$\begin{aligned}
\omega_{1R} &= S_{1R},\\
P_{1R,1} &= \{S_{1R} \to e, e \to de, e \to c\}.
\end{aligned}$$

The initial string of the G_m component be ω_m and the associated tables are as follows:

$$\begin{aligned}
\omega_m &= S_m,\\
P_{m,1} &= \{S_m \to S_m, S_m \to Q_{s1L}, q_3 \to Q_c, x \to F | x \neq q_3\},
\end{aligned}$$

The initial string of the G_c component be ω_c and the associated tables are as follows:

$$\begin{aligned}
\omega_c &= S_c,\\
P_{c,1} &= \{S_c \to S_c, S_c \to Q_{1R}, x \to \lambda, e \to F | x \in (c, d)\},
\end{aligned}$$

In the case of P systems, we denote the current states as follows: (u, v, w), where u denotes the contents of s, v denotes the contents of $1L$ and w denotes the contents of $1R$. To simplify the example, static regions are not marked separately, since their content is always constant. With this triple, we only consider those regions whose contents change.

The initial configuration is based on the construction of the P system, we can choose between two rules (rule 2 and rule 3) for the object a in $1L$ during the first step. Consequently, there are two different cases. Let's examine both cases. First, let's look at an example where Π applies $r_2 : a \to a$ during steps 1 and 2, and the rule 3 .

$$\begin{aligned}
&(bba, a, e) \Rightarrow^{(1,2,7)} (bbe, a, de) \Rightarrow^{(-,2,7)}\\
&(bbe, a, dde) \Rightarrow^{(-,3,8)} (bbe, b, ddc) \Rightarrow^{(1,4,-)}\\
&(ddcddce, d, ddc) \Rightarrow \ldots
\end{aligned}$$

The triple index above the arrows are the numbers of the currently applied rules in each region. For example: $(bba, a, e) \Rightarrow^{(1,2,7)} (bbe, a, de)$ means, that we used rule 1 in s, rule 2 in $1L$ and rule 7 in $1R$.

So we start with (bba, a, e), apply $r_1 : a \to e$ rule in s, rule 2 in $1L$ and rule 7 in $1R$. After applying these rules, the regions change as (bbe, a, de). At this point we are not able to apply rule 1 in s, but we can use rule 2 or rule 3 in $1L$, rule 7 or 8 in $1R$. Use rule 2 and rule 7: (bbe, a, dde). We (still) cannot apply rule 1 in s, apply rule 3 in $1L$ and rule 8 in $1R$: (bbe, b, ddc). Apply rule 1 in s, rule 4 in $1L$: $(ddcddce, d, ddc)$.

In the case of PC ET0L systems, we denote the current strings of components as follows: $(u_{s1L}, u_{1R}, u_m, u_c)$, where u_{s1L} denotes the sentential form of G_{s1L}, where s index denotes the content of S in Π and $1L$ index denotes the content of $1L$ region in Π, u_{1R} denotes the sentential form of G_{1R}, u_m denotes the sentential form of G_m, u_c denotes the sentential form of G_c.

We simulate the steps performed by the polymorphic P system:

$$
\begin{aligned}
&(\ bbaq_0, S_{1R}, S_m, S_c) \Rightarrow^{(1,1,1,1)} (bbQ_{1R}q_0, e, S_m, S_c) \Rightarrow_{com} \\
&(\ bbeq_0, e, S_m, S_c) \Rightarrow^{(1,1,1,1)} (bbeq_0, de, S_m, S_c) \Rightarrow^{(1,1,1,1)} \\
&(\ bbeq_1, dde, S_m, S_c) \Rightarrow^{(2,1,1,1)} (Q_{1R}Q_{1R}eq_2, ddc, S_m, S_c) \Rightarrow_{com} \\
&(\ ddcddceq_2, ddc, S_m, S_c) \Rightarrow \dots
\end{aligned}
$$

The quadruple index above the arrows (except for arrows for communication steps) are the indexes of the currently applied tables in each component. For example: $(bbaq_0, S_{1R}, S_m, S_c) \Rightarrow^{(1,1,1,1)} (bbQ_{1R}q_0, e, S_m, S_c)$ means, that we used the first tables in all components.

Start with $(bbaq_0, S_{1R}, S_m, S_c)$, apply $a \to Q_{1R}$ and $q_0 \to q_0$ rules from $P_{s1L,1}$ in G_{S1L}, $S_{1R} \to e$ rule from $P_{1R,1}$ in G_{1R}, and $S_m \to S_m$, $S_c \to S_c$ from $P_{m,1}$ and $P_{c,1}$ in G_m an G_c (apply these rules until the end of the calculation). With the appearance of the Q_{1R} query symbol, a communication step follows ($\Rightarrow_{com}$ denotes the communication steps). In the communication step, with the help of Q_{1R}, we can insert the sentential form from G_{1R} into G_{s1L}.

Following the further steps, it can be seen that the same values appear in component G_{s1L} as in the s region in Π_2.

With the help of query symbols, PC ET0L can simulate the rewriting of the contents of s in the P system in one step + one communication step.

Let's look at an example where the P system applies rule 2 during step 1 and rule 3 during step 2, i.e. $r_2 : a \to a$ first, and then $r_3 : a \to b$.

$$
\begin{aligned}
&(\ bba, a, e) \Rightarrow^{(1,3,7)} (bbe, b, de) \Rightarrow^{(1,4,7)} (dedee, d, dde) \Rightarrow^{(1,5,7)} \\
&(\ ddeeddeee, d, d^3e) \Rightarrow^{(1,5,8)} (d^3ed^3eeed^3ed^3eeee, d, d^4e) \Rightarrow \dots
\end{aligned}
$$

Similar to the previous example, by taking one step + one-communication steps, the PC ET0L simulates the steps of the polymorphic P system.

$(bbaq_0, S_{1R}, S_m, S_c) \Rightarrow^{(1,1,1,1)} (bbQ_{1R}q_1, e, S_m, S_c) \Rightarrow_{com}$
$(bbeq_1, e, S_m, S_c) \Rightarrow^{(2,1,1,1)} (Q_{1R}Q_{1R}eq_2, de, S_m, S_c) \Rightarrow_{com}$
$(dedeeq_2, de, S_m, S_c) \Rightarrow^{(3,1,1,1)} (Q_{1R}eQ_{1R}eeq_2, dde, S_m, S_c) \Rightarrow_{com}$
$(ddeeddeeeq_2, dde, S_m, S_c) \Rightarrow^{(3,1,1,1)}$
$(Q_{1R}Q_{1R}eeQ_{1R}Q_{1R}eeeq_2, ddde, S_m, S_c) \Rightarrow_{com}$
$(d^3ed^3eeed^3ed^3eeeeq_2, ddde, S_m, S_c) \Rightarrow \ldots$

As we can see in the previous two examples, we can assuem that PC ET0L can simulate the first few steps of the Polymorphic P system in the subsequent steps using the appropriate tables.

In general the configuration of the polymorphic P system can be denoted by a triple, where α is the content of S, x is the content of $1L$, a single object due to the non-cooperative property, and β' is the content of $1R$:

(α, x, β') where $\alpha = \alpha_1 x \alpha_2 x \ldots x \alpha_k$
then we can denote the triple as follows: $(\alpha_1 x \alpha_2 x \ldots x \alpha_k, x, \beta')$.

In general the configuration of the PC ET0L system can be denoted with a 4-tuple, with its components. The sentential form of the first component: u_{s1L}, in which s index means that this component contains the content of s and $1L$ index means that this component also contains the content of $1L$. The sentential form of the second component: β, where β is the ancestor of β' appearing in the polymorphic P system. The sentential form of the third component: S_m, the content of G_m, and the sentential form of the fourth component: S_c, the content of G_c

$(u_{s1L}, \beta, S_m, S_c)$ where $u_{s1L} = \alpha_1 x \alpha_2 x \ldots x \alpha_k q_i$,
then the four component can be denoted as follows:
$(\alpha_1 x \alpha_2 x \ldots x \alpha_k q_i, \beta, S_m, S_c)$ where

the x's are denotes the first components of q_i (i.e., a, b, d, or g_f).

In the case of this general construction, examine how the content of the P system changes after one step. Apply the first rule, which is $x \rightarrow \beta'$, to the content of s, i.e., the first element of the triple. Apply a rule to x from the applicable rules of $1L$, if it exists, let x's successor be x'. Similarly for β', apply a rule from the applicable rules of $1R$, if it exists, let β''s successor be β'':

$$(\alpha_1 x \alpha_2 x \ldots x \alpha_k, x, \beta') \Rightarrow (\alpha_1 \beta' \alpha_2 \beta' \ldots \beta' \alpha_k, x', \beta'').$$

In order for the PC ET0L system to be able to simulate this step, it must take one step and one communication step. In the first component, applying the rule corresponding to x, it rewrites the x's to the query symbols Q_{1R}. In parallel, apply one of the following applicable rules to q_i , the same one that the polymorphic P

system applied to $1L$. In the second component, apply one of the following applicable rules to β, the one that transforms β'. In the third and fourth component, apply $S_m \to S_m$ and $S_c \to S_c$ rules. This step is followed by the communication step. In accordance with the PC ET0L, the content of the corresponding component, in this case the content of the second component, G_{1R}, is copied to the place of the query symbols in the first component. The content of the second component is β', which is the same as the right side of rule 1 applied in the polymorphic P System. It can be seen that the content of the first component of Γ $(\alpha_1\beta'\alpha_2\beta' \dots \beta'\alpha_k q_i')$ is the same as the content of s and $1L$ after one step in the P system:

$$
\begin{aligned}
&(\ \alpha_1 x \alpha_2 x \dots x \alpha_k q_i, \beta, S_m, S_c) \Rightarrow \\
&(\ \alpha_1 Q_{1R} \alpha_2 Q_{1R} \dots Q_{1R} \alpha_k q_i', \beta', S_m, S_c) \Rightarrow_{com} \\
&(\ \alpha_1 \beta' \alpha_2 \beta' \dots \beta' \alpha_k q_i', \beta', S_m, S_c).
\end{aligned}
$$

The polymorphic P system reaches a halting state when there are no rules in any region that can be applied to its content. In this example, the system reaches the halting configuration when there are no applicable rules for the contents of $1L, 1R$ and s.

In $1L$ region, the computation stops, after applying rule $(d \to g_f)$. In $1R$ region, the computation stops, after applying rule $(e \to c)$.

After applying $(d \to g_f)$ in $1L$, rule 1 can never be applied in s again.

In general, we can say that the polymorphic P system is in a halting state if its configuration:

$$(\alpha, g_f, d \dots dc).$$

Then α is the word (multiset) generated by the polymorphic P system.

To generate the α with the PC ET0L system we need G_m and G_c components. The G_m and G_c components apply the corresponding $S_m \to S_m$, $S_c \to S_c$ rules as long as the calculation is in progress. In order to generate α, the rule $S_m \to Q_{S1L}$ must be applied in G_m if the symbol q_3 appeared in G_{S1L}. In parallel, we apply the rule $S_c \to Q_{1R}$ in G_c:

$$
\begin{aligned}
&(\ \alpha q_3, \beta, S_m, S_c) \text{ where } \beta = dd \dots dc \\
&(\ \alpha q_3, \beta, S_m, S_c) \Rightarrow (\alpha q_3, \beta, Q_{S1L}, Q_{1R}) \Rightarrow_{com} \\
&(\ \alpha q_3, \beta, \alpha q_3, \beta) \Rightarrow \dots .
\end{aligned}
$$

Then, after the communication step, the rules of the $P_{m,1}$ table belonging to the G_m component and the $P_{c,1}$ table belonging to the G_c can be applied.

In G_c, $c \to \lambda$ and $d \to \lambda$ rules, delete all c and d letters. If the rules $S_m \to Q_{s1L}$, $S_c \to Q_{1R}$ were applied at the appropriate time of the calculation, all values of the G_c component will be deleted by these rules.

It is important to delete only those letters for which there is no rule, i.e., they will no longer change.

If the process stops at an inappropriate time, then at least one letter e must remain in 1R, that cannot be deleted, and apply the rule $e \to F$, which results in an error.

After the deletions, apply the $q_3 \to Q_c$ in the G_m component; if there is x in α where $x \neq q_3$, then apply the $x \to F$ rule. A trap letter will appear, indicating that the calculation was incorrect.

If the letter F does not appear and the entire content of the G_c component has been deleted, then after the communication step, the content of the G_m component (the master) will be α, which is an accepted word consisting of terminal letters.

$$(\alpha q_3, \beta, \alpha Q'_0, \lambda) \Rightarrow_{com}$$
$$(\alpha q_3, \beta, \alpha, \lambda).$$

4 Conclusion

Based on the example developed in section 3, we can assume that, using similar methods, we can create a PC ET0L system that can simulate its operation for other, even more complex, deeper non-cooperative polymorphic P systems. Our work is an initial step in finding the relationship between non-cooperative polymorphic P systems and parallel communicating ET0L systems.

Acknowledgements

Supported by the University of Debrecen Scientific Research Bridging Fund (DETKA).

References

1. Alhazov, A., Ivanov, S., Freund, R.: Polymorphic P systems: A survey. Bulletin of the International Membrane Computing Society **2** (2016) 79–101
2. Alhazov, A., Ivanov, S., Rogozhin, Y.: Polymorphic P systems. In Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A., eds.: Membrane Computing. Volume 6501 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (2011) 81–94
3. Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Springer-Verlag, Berlin Heidelberg (1997)
4. Păun, G.: Membrane Computing: An Introduction. Springer-Verlag, Berlin, Heidelberg (2002)
5. Păun, G., Rozenberg, G., Salomaa, A., eds.: The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)

6. Lindenmayer, A.: Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. Journal of Theoretical Biology **18**(3) (1968) 280–299
7. Păun, G.: Parallel communicating systems of L systems. In: Lindenmayer systems: impacts on theoretical computer science, computer graphics, and developmental biology. Springer Science and Business Media, Berlin, Heidelberg (1992) 405–418
8. Kuczik, A., Vaszil, G.: Simple variants of non-cooperative polymorphic P systems. Journal of Membrane Computing (to appear)
9. Vaszil, G.: On parallel communicating lindemayer systems. In Păun, G., Salomaa, A., eds.: Grammatical Models of Multi-Agent Systems. Volume 8 of Topics in Computer Mathematics., Gordon and Breach Science Publishers (1999) 99–112

Spiking neural P systems with colored spikes and multiple channels in the rules (extended abstract)

Rodrigo Llanes, José M. Sempere[1]

Valencian Research Institute for Artificial Intelligence (VRAIN)
Universitat Politècnica de València
rodllala@inf.upv.es jsempere@dsic.upv.es

We propose SN P systems that allows the sending/receiving of different kinds of spikes through different channels. Our proposal is related to [5], where the authors proposed the use of different spiking channels to connect the neurons in the network. Observe that the idea of connecting different target neurons in the same rule was initially proposed in [1] where the inclusion of target neurons in the rules is allowed. This idea, and the relationship with multiple channels, was highlighted in [7] where the authors considered this ingredient in the formal framework of SN P systems. The other variant that we have considered in this work is showed in [6], where the authors proposed the use of a non-singleton alphabet to define the spikes of the system.

Here, we combine multiple channels in the neuron rules and different kinds of spikes together. The rules can manage different spikes at every moment, and they can be sent by different channels. This combination of ingedients allows the simulation of other models proposed in the membrane computing research area, such as virus machines [3] and neural-like P systems with plasmids [2].

SN P systems

Definition 1. [4] A spiking neural P system (SN P system, for short) of degree $m \geq 1$ is defined by the following tuple $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where:

1. $O = \{a\}$ is a singleton alphabet of spikes
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where
   a) $n_i \geq 0$ is the initial number of spikes contained in σ_i.
   b) R_i is a finite set of rules in one of the following two forms:
      - firing or spiking rules: $E/a^c \rightarrow a; d$ where E is a regular expression over O, and $c \geq 1$, $d \geq 0$ are integer numbers. We omit E whenever it be equal to a^c, and we omit d whenever it be equal to 0.

[1] Corresponding author

- forgetting rules: $a^s \to \lambda$, for $s \geq 1$, with the restriction that for each spiking rule $E/a^c \to a; d$ then $a^s \notin L(E)$ ($L(E)$ is the regular language defined by E)

3. $syn \subseteq \{1, 2, \ldots m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of synapses between neurons
4. $in,\ out\ \in \{1, \ldots, m\}$ indicate the input and the output neurons of Π.

□

Definition 2. [5] A spiking neural P system with multiple channels (SNP-MC system, for short) of degree $m \geq 1$ is defined by $\Pi = (O, L, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, out)$, where:

1. $O = \{a\}$ is a singleton alphabet of spikes
2. $L = \{1, 2, \ldots, N\}$ is the alphabet of channel labels
3. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (n_i, L_i, R_i)$, $1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the initial number of spikes contained in σ_i.
   b) $L_i \subseteq L$ is a finite set of channels labels used in the neuron
   c) R_i is a finite set of extended rules in the form $E/a^c \to a^p(l)$ where E is a regular expression over O, and $c \geq 1,\ p \geq 0, l \in L_i$.
4. $syn = \{(i, j, l)\} \subseteq \{1, 2, \ldots m\} \times \{1, 2, \ldots, m\} \times L$ with $(i, i, l) \notin syn$ for $1 \leq i \leq m$ and $l \in L$ (synapse connections);
5. $out\ \in \{1, \ldots, m\}$ is the output neuron.

□

In this case, the forgetting rules can be established as those extended rules with $p = 0$ and no regular expression E. Every rule indicates the channel that the rule uses to send the spikes. The spikes are only sent to the connected neurons by the channel established in the rule. In each computation step there can be some competition between different rules at the neuron. Provided that more than one rule can be applied, then the system selects only one of them non-deterministically. So, the neurons work in sequential manner (they apply only one rule at every computation step).

Definition 3. [6] A spiking neural P system with colored spikes (SNP-CS system) of degree $m \geq 1$ is defined by $\Pi = (C, O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where:

1. $C = \{1, 2, \ldots g\}$ is a finite set of colors to mark the color of a spike. Every spike is associated with a unique color
2. $O = \{a_1, a_2, \ldots, a_g\}$ is an alphabet of g colored spikes
3. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (< n_1^i, n_2^i, \ldots, n_g^i >, R_i)$, $1 \leq i \leq m$, where:
   a) $n_h^i \in \mathbb{N}$ is the number of spikes a_h initially placed in neuron σ_i.
   b) R_i is a finite set of spiking and forgetting rules in the form:
      - Spiking rule: $E/a_1^{c_1} a_2^{c_2} \ldots a_g^{c_g} \to a_1^{p_1} a_2^{p_2} \ldots a_g^{c_g}; d$ where E is a regular expression over O, and $c_i, p_i \geq 0$;

- Forgetting rule: $a_1^{s_1}a_2^{s_2}\dots a_g^{s_g} \to \lambda$ where $a_1^{s_1}a_2^{s_2}\dots a_g^{s_g} \notin L(E)$ for any regular expression E associated with a spiking rule;

4. $syn \subseteq \{1,2,\dots m\} \times \{1,2,\dots,m\}$ with $(i,i) \notin syn$ for $1 \leq i \leq m$;
5. $in, out \in \{1,\dots m\}$ are the input and the output neurons

□

The work in this model is similar as in the original model described in [4].

Spiking neural P systems with multiple channels in the rules and colored spikes

We introduce a model of SN P systems that combines the use of multiple channels that can be located both in the synaptic connections and in the rules of the neurons themselves, together with a colored spikes alphabet (i.e. a non-singleton alphabet).

Definition 4. A spiking neural P system with multiple channels and colored spikes (SNP-MC-CS system, for short) of degree $m \geq 1$ is defined by $\Pi = (O, L, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in)$, where:

1. $O = \{a_1, a_2, \dots, a_g\}$ is an alphabet with g colored spikes
2. $L = \{1, 2, \dots, N\}$ is an alphabet of channel labels
3. $\sigma_1, \sigma_2, \dots, \sigma_m$ are neurons of the form $\sigma_i = (w_i, L_i, R_i)$, $1 \leq i \leq m$, where:
   a) $w_i \in O^*$ is a string that denotes the initial multiset of spikes in the neuron.
   b) $L_i \subseteq L$ is a finite set of channels labels used in the neuron
   c) R_i is a finite set of rules in the forms:
      - Firing rules: $E/w_c \to w_1\,(l_1)\,w_2\,(l_2)\dots w_n\,(l_n)\,;d$ where E is a regular expression over O, and $w_c, w_i \in O^*$ $1 \leq i \leq n, l_i \in L_i$, $d \geq 0$.
      - Forgetting rules: $w_c \to \lambda$, where $w_c \in O^*$.
4. $syn \subseteq \{1,2,\dots m\} \times \{1,2,\dots,m,out\} \times L$ are the set of synapse connections with $(i,i,l) \notin syn$ for $1 \leq i \leq m$ and $\forall\ l \in L$. Observe that (i, out, l) denotes that the neuron σ_i sends the spikes out to the environment by the channel l.
5. $in \in 1,\dots,m$ is the input neuron. Observe that the input neuron can be omitted whenever the system is a generator.

□

The firing rules $E/w_c \to w_1\,(l_1)\,w_2\,(l_2)\dots w_n\,(l_n)\,;d$ of the neuron σ_i can be applied whenever $\psi_O(x_i) \in \Psi_O(L_E)$, where x_i denotes the spikes in the neuron σ_i, $\psi_O(x_i)$ denotes the Parikh set of x_i, and L_E is the language denoted by the regular expression E. In such a case, the spikes denoted by w_c are removed from the neuron, and the spikes denoted by w_i are sent to the connected neurons (or to the environment) by the channel l_i according to the synaptic connections of the neuron. If the delay $d > 0$, then the neuron is blocked, and it cannot neither send or receive spikes after d computation steps.

The forgetting rules of the neuron σ_i in the form $w_c \to \lambda$, can be applied whenever no firing rule is applicable and $(\forall a_i \in O)\ |w_c|_{a_i} \leq |x_i|_{a_i}$, where x_i

denotes the spikes in the neuron σ_i. In such a case, the spikes denoted by w_c are removed from the neuron.

The system runs in a maximally parallel manner, that is, at every computation step, every neuron that has some applicable rule will apply it. If there are more than one applicable rule, the firing rules will have priority over those of forgetting. If there are more than one applicable firing rule in conflict (they require common spikes) then, only one of them is non-deterministically selected to be applied, the chosen rule will be executed as many times as possible. Observe that, in this case, the system behavior is similar to the one of transition cell-like P systems. If a rule with a delay is activated, the neuron will be blocked just as in SN P systems. If several rules with delays are applied at the same time, the neuron will be blocked for the maximum delay value of the applied rules. Finally, it should be noted that the compution halts when there is no delayed neuron, and no rule can be applied in any neuron. Regarding the output of the system, we can obtain it in four different ways:

- *halting mode*: The result of the computation is the number of spikes that have been sent to the environment throughout computation, no matter their colors. The output is obtained when the system halts.
- *multi-channel halting mode*: As in the halting mode, the output of the model are the spikes sent to the environment during execution, but they are joined depending on the channels through which they have been sent (i.e. a non-negative integer vector is obtained).
- *temporary mode*: This mode is the one used for generating systems, since it is not necessary reach a halting configuration, the system output will be read as the sequence of spikes sent to the environment at every computation step (a spike train).
- *multi-channel temporary mode*: It can be used for signal processing, as it allows reading the output as the sequence of spikes (a spike train) sent out to the environment at every computation step for every channel.

Theorem 1. SNP-MC-CS systems are universal models of computation.

Simulating other computation models.

The proposed model is able of simulating different models that have been previously proposed in the field of membrane computing. In this way, the SNP-MC-CS model aims to be a unifying framework for other alternative models.

Definition 5. [3] A Virus Machine (VM, for short) of degree $l(p,q)$,$p,q \geq 1$, is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ where:

1. $\Gamma = v$ is a singleton alphabet;
2. $H = h_1, \ldots, h_p$ and $I = i_1, \ldots, i_q$ are ordered sets such that $v \notin H \cup I$ and $H \cap I = \emptyset$;

3. $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ is a weighted directed graph, where $E_H \subseteq H \times (H \cup \{h_{out}\})$, $(h,h) \notin E_h$ for each $h \in H$, *out-degree*$(h_{out}) = 0$, and w_H is a mapping from E_H onto $\mathbb{N} - \{0\}$ (the set of positive integer numbers);
4. $D_I = (I, E_I, w_I)$ is a weighted directed graph, where $E_I \subseteq I \times I$, w_I is a mapping from E_I onto $\mathbb{N} - \{0\}$ and, for each vertex $i_j \in I$, the out-degree of i_j is less than or equal to 2;
5. $G_C = (V_C, E_C)$ is an undirected bipartite graph, where $V_C = I \cup E_H$, being I, E_H the partition associated with it. In addition, for each vertex $i_j \in I$, the degree of i_j is less than or equal to 1.
6. $n_j \in \mathbb{N}(1 \leq j \leq p)$;
7. $h_{out} \notin I \cup v$ and h_{out} is denoted by h_0 in the case that $h_{out} \notin H$.

□

Theorem 2. For every Virus Machine there exists an equivalent SNP-MC-CS system.

Definition 6.[2] A neural-like P systems with plasmids (NP P system, for short) of degree $m \geq 1$, is a construct of the form $\Pi = (O, b_1, \ldots, b_m, link, in, out)$ where:

1. $O = \{p_1, \ldots, p_w\}$ is an alphabet of *plasmids*
2. $b_1, \ldots, b_m$ are bacteria of the form $b_i = (N_i, R_i)$ for $1 \leq i \leq m$ where $N_i = \langle n_1, n_2, \ldots, n_w \rangle$ is a vector and $n_j \geq 0$ is the initial number of plasmids p_j inside bacterium b_i for $1 \leq j \leq w$; and R_i is a finite non-empty set of plasmid rules of the form $C/r_1, r_2, \ldots, r_n$ where condition C is a multiset over O, and, each r_k for $1 \leq k \leq n$ is either one of two forms:
   a) A *transmit* operation: $P \to out$, where P is a submultiset of C;
   b) A *kill* operation: $P \to \lambda$, where P is a submultiset of C;
3. $link \subset \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i,i) \notin link$ for $1 \leq i \leq m$ (the links between bacteria);
4. in, out indicates the input and output bacterium of the system, respectively. They can be omitted, depending on whether the system is generating outputs or accepting inputs.

□

Theorem 3. For every NP P system there exists an equivalent SNP-MC-CS system.

Conclusions

In this work we have proposed the combination of two variants that had been previously proposed. We propose that the combination of both variants allows the simulation of very diverse models, so the combined model can be viewed as an unification one.

Our future work will be based on considering other models and checking if our proposal is still so general as to be able to simulate them.

References

1. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik (2006) Extended Spiking Neural P Systems. In: H.J. Hoogeboom, G. Păun, G. Rozenberg, A. Salomaa (eds). Membrane Computing. WMC 2006. LNCS vol, 4361. pp 123-134 Springer.
2. F.G.C. Cabarle, X. Zeng, N. Murphy, T. Song, A. Rodríguez-Patón, X. Liu. (2021) Neural-like P systems with plasmids. *Information and Computation*, Vol. 281, 194776.
3. X. Chen, M.J. Pérez-Jiménez, L. Valencia-Cabrera, B. Wang, X. Zeng (2016) Computing with viruses. *Theoretical Computer Science* 623 pp 146-159.
4. M. Ionescu, Gh. Păun, T. Yokomori (2006) Spiking neural P systems. *Fundamenta Informaticae*, 71 pp 279-308
5. H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Lou. (2017) Spiking neural P systems with multiple channels. *Neural Networks,* Vol. 95, pp 66-71.
6. T. Song, A. Rodríguez-Patón, P. Zheng, X. Zeng (2018) Spiking Neural P Systems With Colored Spikes. *IEEE Transactions on Cognitive and Developmental Systems* 10(4) pp 1106-1115.
7. S. Verlan, R. Freund, A. Alhazov, S. Ivanov, L. Pan (2020) A formal framework for spiking neural P systems. *Journal of Membrane Computing* Vol. 2, pp 355–368.

Integrating Human Behavior and Membrane Computing in Epidemiological Models

Fareed Muhammad Mazhar[1], Davide Valcamonica[1], Alberto d'Onofrio[3], Giuditta Franco[2], Claudio Zandron[1]

[1]Department of Computer Science, Systems, and Communications, Università degli Studi di Milano-Bicocca, Milan, Italy
[2]Department of Mathematics, Informatics and Geosciences, University of Trieste, Italy
[3]Department of Computer Science, School of Science and Engineering, Università degli Studi di Verona, Verona, Italy

Summary. The intersection of human behavior and epidemiology is a focal point for understanding infectious disease dynamics, particularly highlighted by epidemic outbreaks such as COVID-19. This research introduces an epidemiological model, derived from the principles of membrane computing, then inspired by biological processes, to analyze the intricate interplay between societal behavior and disease transmission. The model, structured hierarchically with Eco-Membranes, Province-Membranes, and Place-Membranes, facilitates the simulation of diverse geographical and social environments, capturing the complex dynamics of infectious diseases within various demographic segments. In this innovative approach lies the integration of mathematical functions to articulate societal responses to infection rates and vaccination willingness. These functions are based on the ratio of infected individuals to the total population, vaccine efficacy, and duration data sourced from multiple studies. The inclusion of demographic characteristics, societal behaviors, response to infections, and vaccination dynamics provides a multi-dimensional view of disease spread, especially under the lens of the COVID-19 pandemic. Through comprehensive simulations, the model examines scenarios incorporating different behavioral responses and intervention strategies, including vaccination dynamics. Sensitivity analysis confirms the robustness of the model, revealing the critical parameters that influence the spread of the virus, thus providing valuable insights for targeted public health interventions.

Keywords: Epidemiological Modeling, Membrane Computing, Behavioral Epidemiology, Infectious Diseases, COVID-19, Vaccination Dynamics.

1 Introduction

In epidemiology of infectious diseases, the emergence of objection to vaccination against some diseases such as measles and the ongoing COVID-19 pandemic [1] has emphasized the critical need for innovative and adaptable modeling approaches. Traditional models [2], while providing valuable insights, often fall short in captur-

ing the complex interplay between human behavior and disease spread dynamics [3]. To address this limitation, a new discipline known as Behavioral Epidemiology of Infectious Diseases has emerged [3, 4], aiming to integrate social science concepts with infection transmission models [3, 4]. This facilitates a deeper understanding of the complex interplay between human behavior and disease spread.
The present study employs the paradigm of membrane computing [5], inspired by biological processes, to offer a new perspective on behavioral epidemiology modeling. By employing a hierarchical structure comprising Environment, Provinces and places are modeled by membranes, this model enables detailed simulation of geographical areas and public places. Each Province-Membrane encompasses Place-Membranes representing specific locations such as schools, workplaces, hospitals, and common areas, while objects within the simulation environment denote elements like time indicators and individual characteristics. Central to our model is the societal behavior response to infections, the vaccination dynamics and the incorporation of demographic characteristics. Our approach provides insights into disease spread dynamics, particularly in the context of COVID-19. The model explicitly includes societal behavior responses to infection rates and vaccination willingness, considering factors like the ratio of infected individuals to the total population. Additionally, data on vaccine effectiveness [6] and duration from various sources are incorporated to realistically model infection spread [7]. By incorporating infection rules, evolution dynamics, and daily routines for different demographic groups, the model provides a comprehensive framework for modeling infection dynamics and daily behaviors within various scenarios. Overall, this study explores the application of Membrane Systems to epidemiological research, aiming to develop an integrated behavioral epidemiology model that accurately represents infectious disease transmission dynamics, and intervention strategies while considering the influence of human behavior.

2 Model Description in P System

The proposed model employs a hierarchical structure consisting of Eco-Membrane, Province-Membranes, and Place-Membranes, facilitating detailed simulation of geographical areas or public places. This hierarchical setup enables to corectly represent the intricate interplay of locations and activities that are central to an individual's everyday routine. This approach allows for the simulation of spatial and social dynamics pertinent to disease spread. Specifically, within each province membrane, key establishments such as schools, workplaces, hospitals, and common areas transitional spaces connecting different regions are delineated [8] by modelling them as Place-Membranes contained in the Province-Membranes. Objects represent elements within the simulation environment such as time indicators and individual characteristics. Key objects are included as; Hour ($Hour_i$), Infection Number (ϕ), Day (d_i), and demographic categories like Young (g), Adult (a), and Elderly (an). The role of human behavior in modulating (e.g. by means of spontaneous and forced social distancing) is embedded by means of appropriate function

$\Psi(M)$ that describes societal behavior response to infection rates, where M is an information index [3, 4] modeling the information individual has concerning the spread of the disease.

As far as the disease control, in the model is included the possibility that a vaccination campaign is enacted as well as the possible partial adherence to the campaign due to objection to vaccination. The model introduces a function $\omega(M)$ to represent willingness to get vaccinated, considering as information index the ratio of infected individuals to the total population [3, 4]. Vaccine effectiveness and duration of the immunity given by the vaccine are incorporated to model infection reduction realistically.

The LOIMOS framework [1] categorizes infection rates based on immunity and symptoms, applied across various environments like common areas, schools, workplaces, etc. Rules consider factors like day, time, infection probability, vaccination status, and mask usage. Infection probability is calculated on the base of current infections, total individuals, and a decreasing function modeling contagiousness. Virus incubation lasts for 05 days after contact, transitioning individuals to an infected state. After incubation, infection progresses through 07 days, followed by recovery. Recovery grants natural acquired immunity, akin to a perfect vaccine, lasting for 180 days. Daily schedules are outlined for young individuals, workers, and the elderly, including activities in common areas, schools, workplaces, and homes. Rules govern movement between locations, such as entering schools or workplaces and returning home. Elderly individuals engage in tasks in common areas with probabilities for different durations and return home afterward. These rules and routines provide a comprehensive framework for modeling infection dynamics, evolution, and daily behaviors within the scenario, incorporating various factors contributing to disease spread and progression. We built our model and software by adapting the LOIMOS framework [1], namely adding mobility and behavioral response. Technical details of the above illustrated membrane system are focused on a submitted paper which is not published yet (however it may be made available upon request): in the following of this work instead we emphasize the simulation work, which advanced aside the theoretical development of the model.

3 Implementation of the model

The implementation of the model is guided by the objective of creating a simulation framework for infectious diseases using P Systems theory, integrating dynamic and behavioral logic into a baseline derived from prior research. The model draws inspiration from, and goes significantly beyond, the work by Baquero and coworkers [1]. The keys aspects in this model are membrane structure forms the basis for organizing a hierarchical structure of a generic epidemiological scenario. This involves dividing a geographical region into Province-Membranes and further subdividing them into Place-Membranes. Entities within the model, such as individuals and contextual resources, are represented using object-oriented logic.

This facilitates the mapping from P Systems to a programming language, treating individuals as singular programming objects with their parameters. A well-defined membrane structure enables the modeling of different compartments of the scenario, organizing relevant aspects of the epidemiological scope efficiently. Rules in the P Systems approach represent computations, encapsulating processes like infection progression, human movement, and vaccination. These rules are translated into methods and functions within the model. The model can extend beyond COVID-19 to other communicable diseases by adjusting parameters related to contagion and infection progression. New behavioral logics, demographic information, and intervention strategies can also be incorporated.The model is scaled to represent larger scenarios by adding more individuals and expanding the geographic scope. The simulation aspect can also scale up to handle larger computational loads by parallelizing the simulation process. The foundation laid by these works informs the development of a comprehensive epidemiological model based on P Systems. The model's adaptability and scalability enable it to evolve beyond its initial scope, accommodating new disease parameters, variations in agent behavior, and different intervention strategies. Additionally, the membrane structure and objects within the model are defined through specialized classes. Membranes such as schools, workplaces, hospitals, and common areas are represented, each containing individuals as instances of the Individual class. Attributes and functionalities of Province-Membrane and Place-Membrane classes facilitate the management of geographical regions and local environments, while the Individual class encapsulates attributes and functionalities relevant to individual agents. Behavioral logic, integrated with vaccination logic for convenience, calculates factors such as caution and vaccination willingness based on the current epidemiological situation. Functions within the Behavioral Logic class handle the assignment of vaccine effectiveness and correlate it with duration. The model's design is grounded in theoretical concepts from P Systems theory and informed by empirical findings [6]. Through carefully the implementation of membrane structures, objects, and behavioral logic, the model provides a versatile framework for simulating and analyzing infectious disease dynamics. Sensitivity analysis plays a pivotal role in unveiling the parameters that significantly influence the dynamics of virus transmission, thus offering invaluable insights for crafting targeted intervention strategies [4]. By focusing on parameters identified as having high impacts through both local and global sensitivity analyses, public health officials can enhance their efforts in mitigating the spread of infectious diseases [9]. This approach is especially critical in overcoming the hurdles posed by data collection challenges and uncertainties surrounding parameter values. The combined insights from sensitivity analysis and the developed models shed light on the multifaceted nature of viral transmission, emphasizing the necessity of addressing multiple transmission pathways and environmental influences. Together, they form a comprehensive framework that not only aids in understanding the complexity inherent in the spread of viral infections but also serves as a cornerstone for forecasting outbreak patterns and formulating effective public health responses. This integrated perspective underscores the critical

need for a holistic approach to disease control, encompassing a broad spectrum of interventions and a deep understanding of the disease ecology. The software is designed to model the spread and control of infectious diseases by simulating various epidemiological scenarios. It focuses on recognizing and validating dynamic patterns of infectious diseases and providing evolutionary predictions in various scenarios. The model is structured to utilize Membrane Systems, which are inspired by the functioning of biological cells. It incorporates hierarchical representation of environments, describes the movement of individuals in these environments, and models biological processes like the incubation and infection of viruses [1]. The model aims to provide a framework for better understanding the dynamics of infectious diseases through simulation results and validating the predictive capacity of the model against given scenarios.

4 Validations of the Results

The effectiveness of the model in predicting the dissemination of infectious diseases has been assessed, focusing on the progression trends of the illness over an extended period, particularly concerning prevalence and mortality rates. Important factors for the validation phase are highlighted in the following. This analysis aims to delve into the outcomes of different scenarios, starting from a scenario where there is minimal behavioral response to the disease and no implemented control measures. The objective is to first investigate the epidemic's behavior under these specific conditions and evaluate the model's capability to accurately reflect the patterns of the outbreak. Introducing a few infected individuals into an entirely susceptible population results in an initial rapid increase in cases. Given the scenario of low adherence to preventive measures and the absence of any intervention strategies, the basic reproduction number, R0, is expected to surge beyond 1 or 2 swiftly, affecting most of the population and eventually stabilizing the epidemic. Without incorporating vital dynamics into the model, the infection is predicted to eventually cease.

In Figure 1, the results of this first scenario are reported. The x-axis represents time in days, from day 0 to day 365, covering a full year, while the y-axis represents the prevalence, which is the number of individuals who are currently infected with the disease at any given time. The curve itself peaks relatively early in the time span, with the highest prevalence occurring around day 50. The peak prevalence is shown to be just over 6,500 cases, which is the maximum number of individuals who are simultaneously infected during the outbreak. After the peak, the prevalence rapidly decreases, indicating that the number of new daily infections drops as the population either recovers or succumbs to the disease. The graph returns to near-zero prevalence after the peak, suggesting that the epidemic subsides, and that the disease no longer actively spreads within the population. The graph depicts a rapid, unvarying increase at the outset, leading to a reach of approximately 7000 cases of infection around the 24th day of the model run-through.

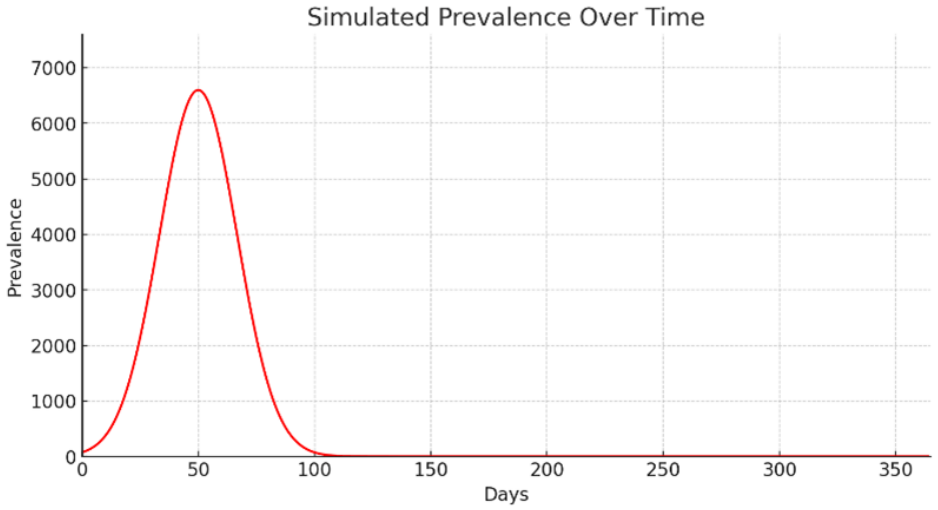


Fig. 1: A simulated epidemic curve showing the prevalence of an infectious disease over time in a population.

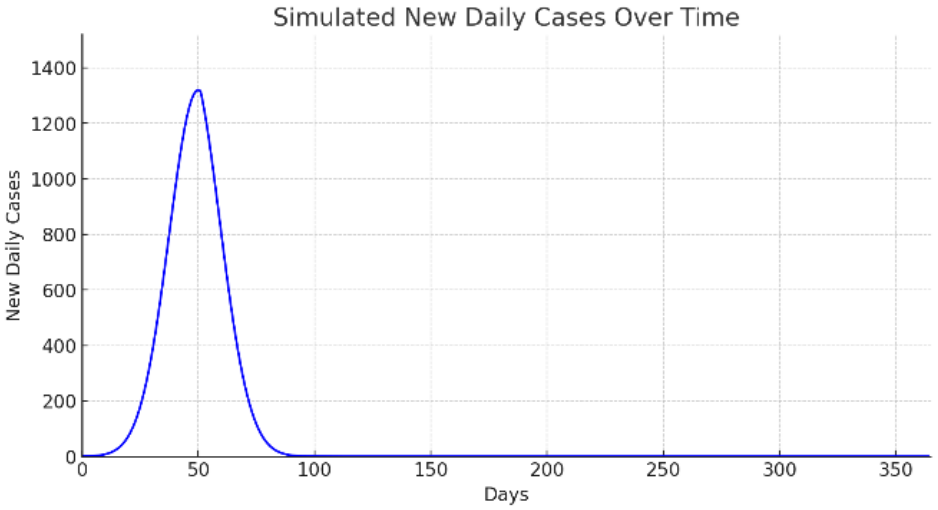


Fig. 2: Trend of new daily cases over the course of a year for an infectious disease outbreak within a population of 30,000 individuals.

In Figure 2, the number of new daily cases are reported, over a one-year period (365 days). The curve spikes sharply, with a peak suggesting that the highest number of new daily cases occurs around day 50. The peak indicates that the

number of new daily cases rises to slightly over 1300. After the peak, the curve shows a steep decline, signifying a rapid drop in the number of new daily cases. Post-peak, the number of new cases gradually approaches zero, suggesting the outbreak is subsiding.

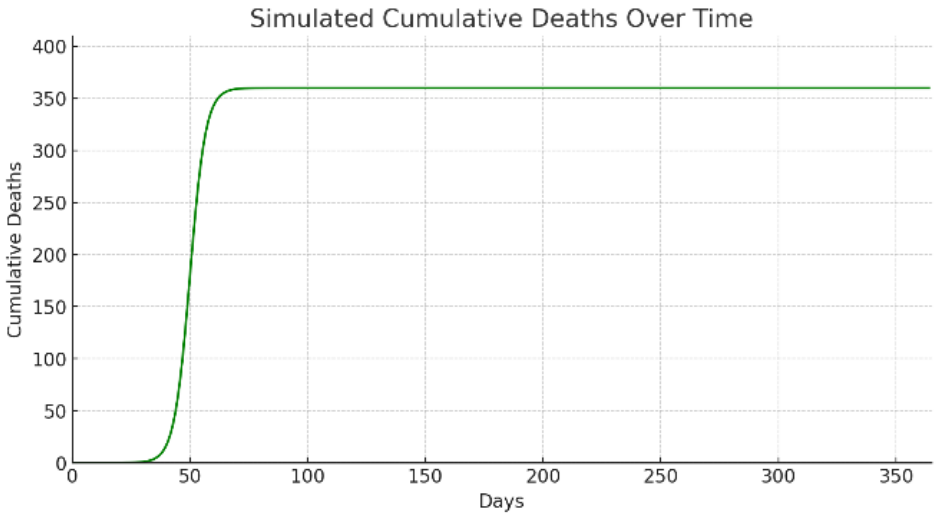


Fig. 3: Cumulative number of deaths over a year in a population of 30,000 individuals during an infectious disease outbreak.

In Figure 3, the y-axis indicates the cumulative number of deaths. The curve shows a rapid rise in deaths early on, reaching a plateau of just over 350 deaths around day 50. Following the step initial increase, the curve flattens, indicating that no further deaths are recorded after reaching the plateau.

Figure 4 reports two graphs. The top graph (New Daily Cases Over Time) marked with blue dots, indicates the number of new cases reported each day. There are two distinct peaks, suggesting two separate waves of infection. The first peak occurs before day 50 and rapidly declines, but not to zero, indicating that the infection was controlled but not eradicated. The second wave starts to rise around day 150 and peaks higher than the first, before declining again. The pattern suggests a relapse or a second outbreak, possibly due to a relaxation of preventive measures or the emergence of a more contagious variant. The bottom graph (Prevalence Over Time) marked with red dots, illustrates the total number of active cases at any given time. Like the new daily cases graph, there are two peaks. The first peak is sharp, indicating a rapid increase in active cases, which then declines rapidly, possibly due to recovery or death of patients. The second peak follows the rise in the new daily cases graph, indicating a second wave of active infections. However, the second prevalence peak is not as sharp as the first, which

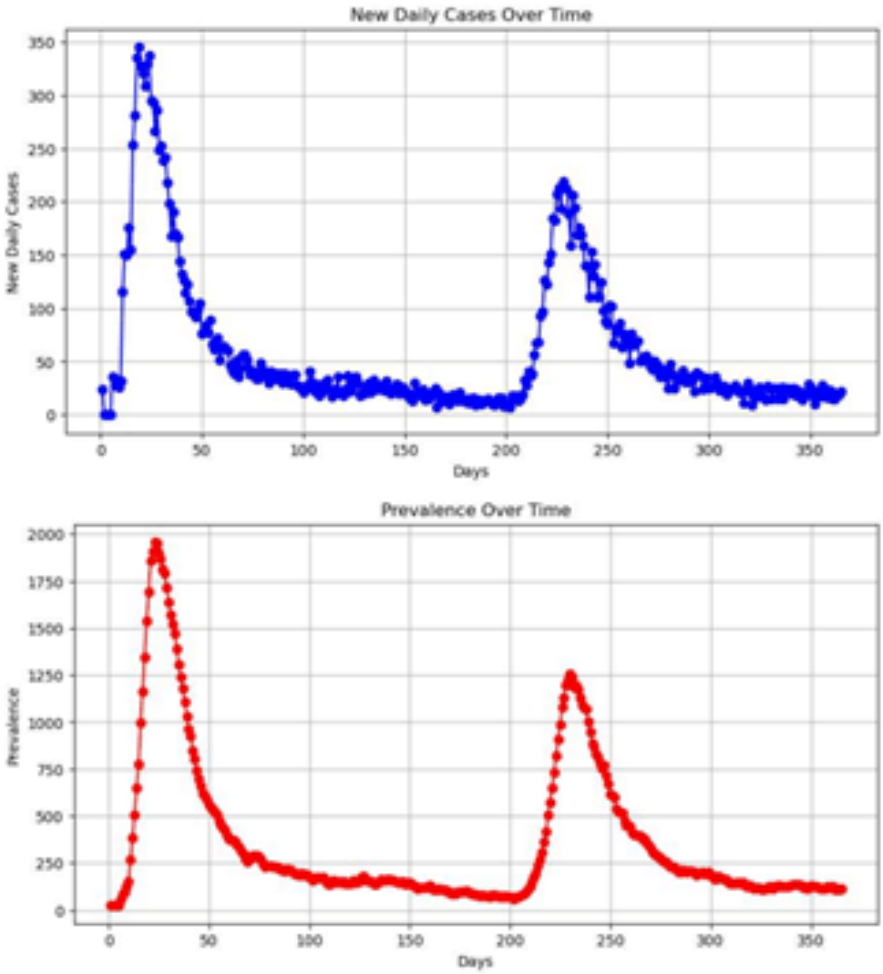


Fig. 4: Progression of an infectious disease over time, measured in days, as seen in two different metrics: new daily cases and prevalence.

may suggest a slower rate of transmission or a more effective response to the second wave. Both graphs together show a disease that has at least two significant periods of transmission. The time between the peaks could indicate successful intervention measures that temporarily contained the spread of the disease, a period of lower transmission rates, or possibly the time it took for the disease to resurge or for a different strain to spread. The graphs do not decline to zero, suggesting the disease continues to persist in the population beyond the timeframe shown. The peaks and troughs of these graphs would be of significant interest in analyzing the

effectiveness of public health interventions, the natural behavior of the disease, and the response of the public to the presence of the disease over time.

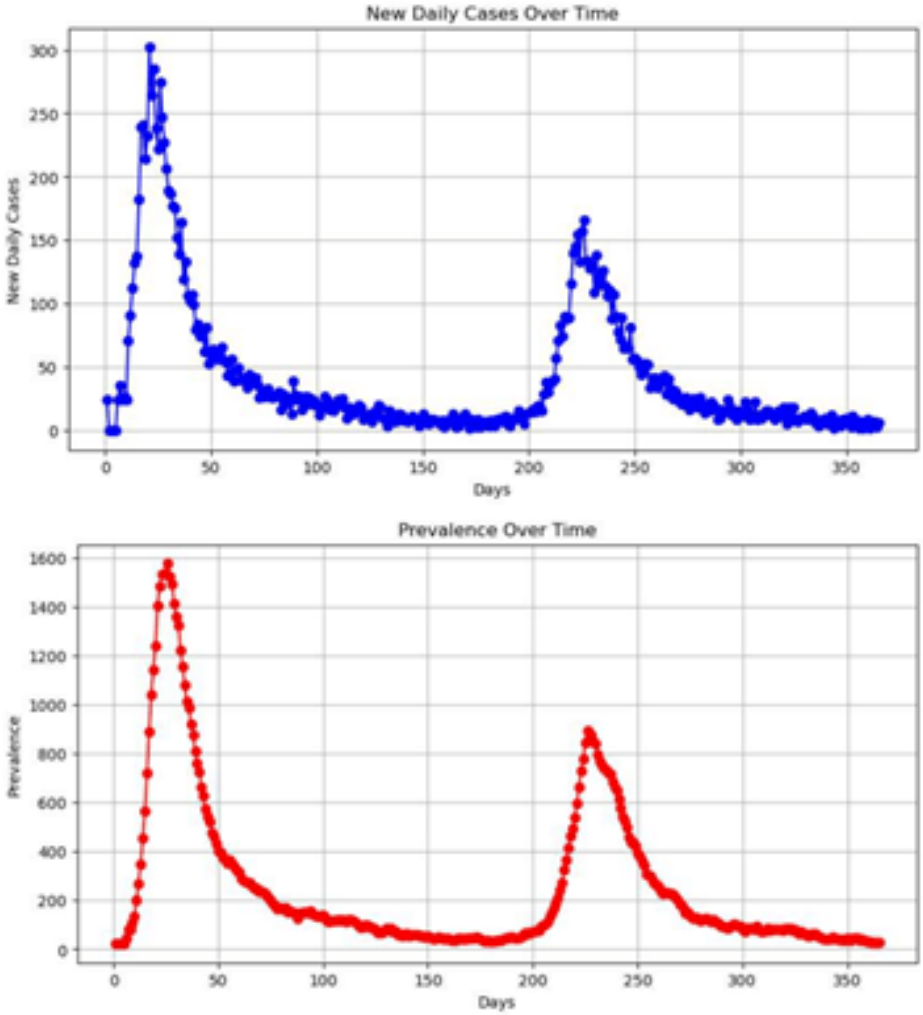


Fig. 5: Epidemiological curves depicting the spread of a disease over time.

In Figure 5, the graph plotted with blue dots shows the number of new cases reported each day. The x-axis represents time in days, and the y-axis represents the number of new daily cases. There are two peaks observed, suggesting two separate waves of the disease. The first peak occurs just before day 50, and the second

peak is around day 225. The error bars on the dots may indicate the variability or uncertainty in the daily case counts.

The second graph (Prevalence Over Time) shows the prevalence of the disease over time. The x-axis is consistent with the first graph, indicating time in days. The y-axis shows the prevalence, which is typically the number of active cases at a given time. Similar to the first plot, two peaks are observed, corresponding to the two waves of new cases. The prevalence peaks are also around day 50 and day 225, shortly after the peaks in new cases. The prevalence curve suggests that as new cases rise, the number of active cases (prevalence) also rises. As new cases drop, so does the prevalence.

In our exploration of disease spread dynamics, we delved into several critical factors that significantly impact the control and progression of infectious diseases. Through simulations, we analyzed the effect of varying levels of vaccination coverage on the temporal spread of disease, providing valuable insights into the efficiency of vaccination campaigns in curbing outbreaks. Additionally, we investigated the influence of the population's behavior, quantified through the caution parameter, on disease transmission. This analysis underscored the profound effect that collective behavioral changes have on slowing the spread of infections. Finally, by comparing infection peaks across scenarios with and without behavioral interventions, we highlighted the tangible benefits of public adherence to recommended preventive measures. These explorations collectively affirm the multifaceted approach needed in managing infectious diseases, emphasizing the synergy between vaccination efforts and public behavior in controlling and eventually overcoming disease outbreaks.

Plot in Figure 6 shows the number of infections over a year with different levels of vaccination coverage (10%, 50%, and 90%). As expected, higher vaccination coverage significantly reduces the peak and spread of infections, demonstrating the importance of vaccination in controlling an outbreak.

Plot in Figure 7 examines the effect of different levels of public caution on disease spread, modeled by caution parameters of 0.5, 1, and 2. A higher caution parameter, indicating increased preventive behaviors by the population, results in a lower peak of infections and a delayed outbreak, highlighting the effectiveness of public health measures and behavioral adjustments.

Plot in Figure 8 compares the progression of the disease in scenarios with no interventions versus those with behavioral interventions, such as increased public caution. It clearly illustrates that behavioral interventions can significantly reduce the peak and overall number of infections, underscoring the critical role of public behavior in managing infectious disease outbreaks.

The model effectively captured the rapid initial surge in both prevalence and new daily infections, subsequently transitioning into a consistent prevalence rate. The mortality pattern echoes that of the prevalence, suggesting a delay between the spike in infections and ensuing deaths. Without any intervention measures, the epidemic achieves equilibrium, with the prevalence stabilizing post the initial swift increase. Cumulatively, these outcomes affirm the model's proficiency in emulating

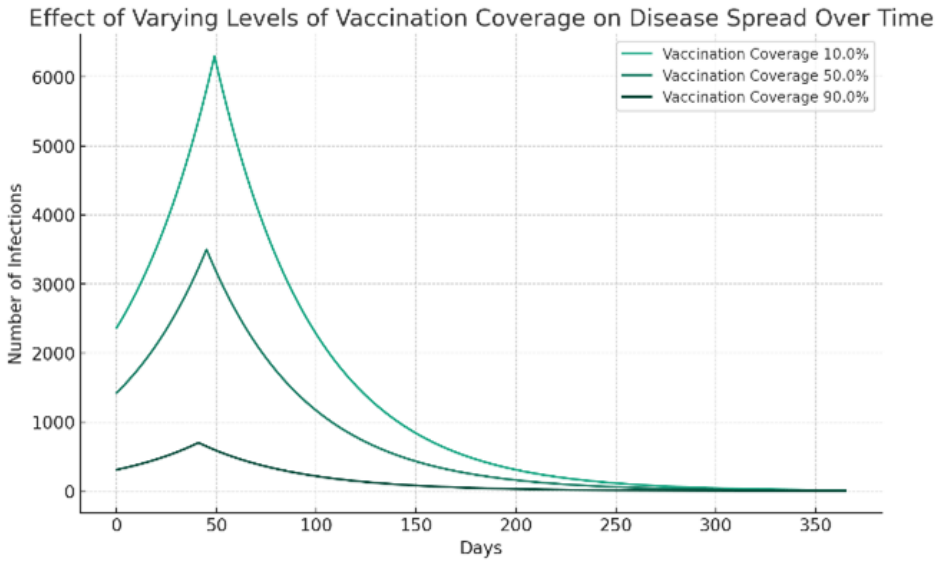


Fig. 6: Effect of Varying Levels of Vaccination Coverage on Disease Spread Over Time.

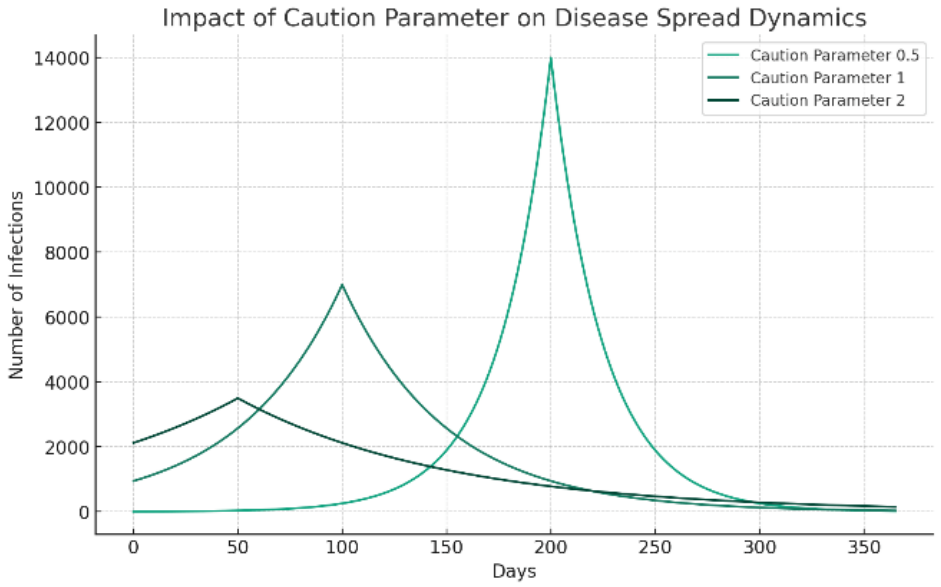


Fig. 7: Impact of Caution Parameter on Disease Spread Dynamics.

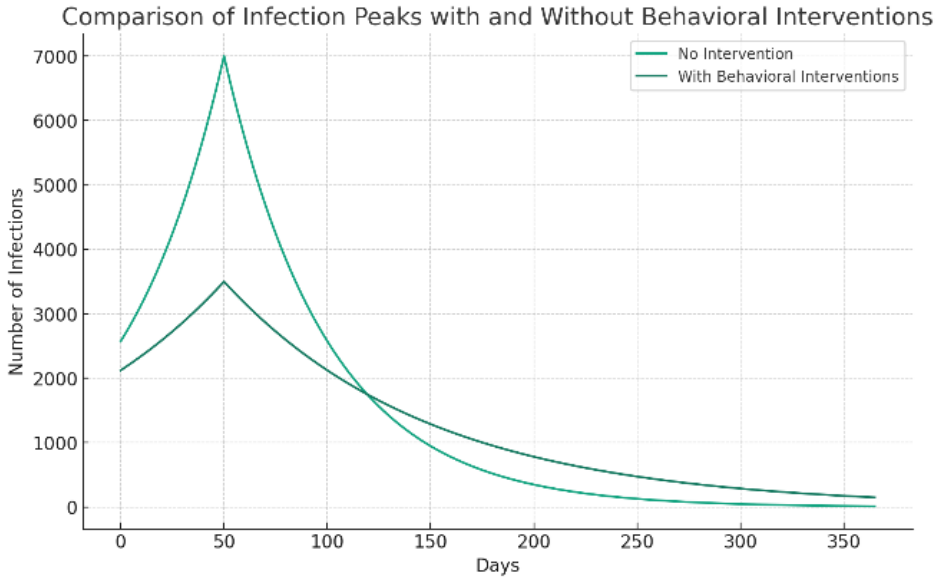


Fig. 8: Comparison of Infection Peaks with and without Behavioral Interventions.

the dissemination of infectious diseases under conditions of minimal behavioral response, and a lack of intervention measures, offering crucial understanding of the epidemics behavior and trajectory.

In the results of Vaccine dynamics, studying these methods to curb virus spread is a critical focus in the field of epidemiology. Among these, vaccination initiatives stand out as key societal measures for halting infectious disease proliferation. Grasping how well these strategies work is essential to evaluate the accuracy and reliability of the epidemiological model.

Graphs in Figure 9 help in understanding how increasing vaccination coverage impacts both the magnitude of these health metrics and the timing of when these maximum values occur. As vaccination coverage increases, there's a clear trend of decreasing max values for all metrics, indicating the effectiveness of vaccination in controlling the disease. Additionally, the timing (day of occurrence) for maximum deaths shifts significantly with higher vaccination coverage, highlighting the changing dynamics of the disease spread and mortality as more of the population becomes vaccinated. The impact of increasing vaccination coverage on the spread of an infectious disease, showing a clear decline in the maximum values of prevalence, new daily cases, and deaths as coverage expands from 10% to 90%. At the lowest coverage, prevalence peaks at 1821 cases by day 21, with new daily cases and deaths reaching their maxima shortly before and much later, respectively. As vaccination coverage grows, not only do these numbers decrease significantly, but the peak days for new cases tend to occur earlier, while for deaths, they shift

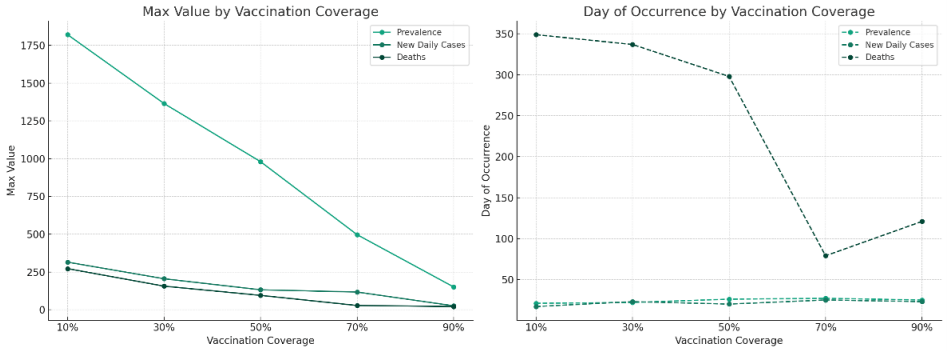


Fig. 9: "Max Value" reached by each metric (Prevalence, New Daily Cases, Deaths) across different vaccination coverage levels (left), and "Day of Occurrence" for the maximum values of each metric, again across varying levels of vaccination coverage (right).

variably. By the time coverage reaches 90%, the maximum prevalence plummets to 151, new daily cases drop to 26, and deaths reduce to 21, demonstrating the efficacy of vaccinations in controlling the outbreak.

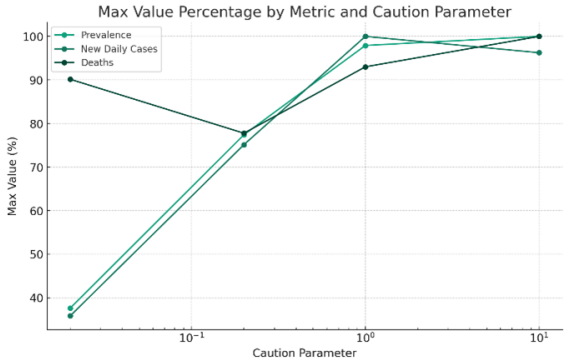


Fig. 10: Relationship between increasing vaccination coverage and its impact on disease metrics within a population of 30,000 individuals.

Plot in Figure 10 shows the "Max Value (%)" for each metric (Prevalence, New Daily Cases, Deaths) across different caution parameters. The x-axis represents the caution parameters on a logarithmic scale to better visualize the wide range of values, while the y-axis shows the maximum value as a percentage of the highest value observed within each metric category. This visualization helps in comparing

the relative changes in prevalence, new daily cases, and deaths as the caution parameter increases.

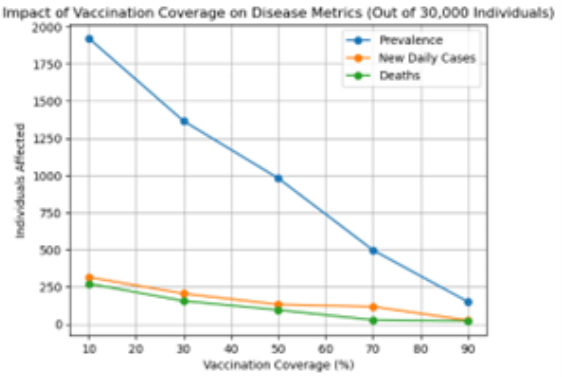


Fig. 11: Relationship between increasing vaccination coverage and its impact on disease metrics within a population of 30,000 individuals.

In Figure 11 a plot concerning the impact of different percentage of the population that has been vaccinated on disease metrics is reported. The data points are spread across five key vaccination coverage milestones: 10%, 30%, 50%, 70%, and 90%. In particular, the maximum number of individuals affected by the disease (Y-axis) plotted for each of the vaccination coverage percentages (X-axis) is represented. It is the count of individuals who are either currently infected (Prevalence), newly infected (New Daily Cases), or have died due to the disease (Deaths) on the day when the maximum value was observed. Prevalence (Blue Line): This line indicates the maximum number of active disease cases at any given point within the population, on the days when the peak prevalence was observed. It shows a clear declining trend, indicating that as vaccination coverage increases, the prevalence of the disease decreases. New Daily Cases (Orange Line): This line tracks the maximum number of new infections reported daily. Similar to prevalence, there is a notable decrease in new daily cases as the vaccination coverage grows. Deaths (Green Line): This line reflects the peak number of deaths recorded in a single day. It also shows a downward trend, which suggests that higher vaccination rates are associated with lower mortality on the day when the maximum deaths occurred. In the context of the case study, the introduction of a "Caution Parameter" enhances the model of disease spread by accounting for human behavior in response to infection rates. This parameter operates through a mathematical formula where $\Psi(f)$ signifies the adjusted probability of infection based on the current fraction of infected individuals, f, relative to a threshold fraction, f*. The threshold fraction, f*, represents a critical level of infection that triggers a heightened level of caution among the population, effectively reducing the infection probability by half.

Essentially, this mechanism transitions a simplistic probabilistic model into a nuanced stochastic one, where the infection dynamics are modulated not just by the raw infection rates but also by the population's adaptive response to the spread of the infection. This adaptive response, governed by the value of f*, implies that a smaller fraction of infected individuals could lead to a significant behavioral shift towards reducing infection risks, thus influencing the overall spread of the disease in a realistic and complex manner.

5 Discussion

Produced data presents the results of a study exploring the impact of varying levels of public caution and vaccine coverage on three key epidemiological metrics: prevalence, new daily cases, and deaths, within a population with no vaccine coverage. The "Caution Parameter" represents a numerical value assigned to the population's level of caution or preventive measures taken to avoid infection. A higher value signifies greater caution and, presumably, more robust preventive behaviors.

As a consequence, at the lowest level of caution (0.02), the impact on disease spread and outcomes is minimal, with relatively high percentages in prevalence (37.64%), new daily cases (35.86%), and particularly high in deaths (90.16%). This suggests that without significant behavioral changes or interventions, the population experiences substantial impacts from the disease.

Increasing the Caution Parameter to 0.2 shows a dramatic increase in all metrics, indicating that even moderate increases in public caution can have a significant effect on disease outcomes. Prevalence and new daily cases rise sharply, indicating a widespread outbreak, but deaths increase at a slower rate (77.78%), suggesting that increased caution might somewhat mitigate the severity of outcomes. At a Caution Parameter of 1, there is a notable shift; while prevalence and deaths increase, with prevalence nearly reaching the entire population and deaths at 93.02%, new daily cases hit 100%. This point might represent a critical threshold where the population caution has a maximized effect on slowing the spread, albeit with a significant portion of the population already affected.

The most extreme caution level analyzed, 10, results in the maximum values for prevalence and deaths, both reaching 100%, while new daily cases slightly decrease to 96.26%. This could indicate a scenario where extreme caution is enacted too late, after the disease has already spread extensively, or it could reflect a situation where extreme caution leads to effective control of new cases, but the overall impact of the disease remains high due to previous spread, and in-between public behavior (as quantified by the Caution Parameter) and disease dynamics in the absence of vaccination. It suggests that while increased caution can significantly affect disease spread and mortality, there is a nuanced balance between the timing and intensity of these behavioral changes and their ultimate impact on disease outcomes. These findings highlight the importance of timely and proportionate public health responses in managing infectious disease outbreaks.

6 Conclusion and future work

This study advances an evidence in the literature about the potential of Membrane Systems in epidemiological modeling, by establishing an integrated framework that aptly represents disease transmission dynamics and intervention effectiveness, considering behavioral influences. The simulation results underscore the necessity of a holistic approach to disease control, which is essential for crafting effective public health strategies in response to infectious disease threats.

A future work could consider the model extension to other infectious diseases and in genereal global scenarios. Indeed, the current model has been extensively applied to the context of COVID-19. A valuable extension would be to adapt and apply this model to other infectious diseases, such as influenza, Ebola, or even antimicrobial resistance, which present different transmission dynamics and societal impacts. Additionally, incorporating geographical variations and cultural differences in human behavior across different global regions could provide insights into disease spread and control measures in a more diversified manner. This expansion would involve adjusting the model parameters to suit different disease characteristics and transmission modes, as well as integrating diverse behavioral responses based on cultural norms. Beside, by utilizing real-time data, such as infection rates, vaccination rates, and public mobility patterns from various sources like health departments and mobile devices, the model could dynamically update and predict disease spread scenarios more accurately.

Acknowledgments

The work of Muhammad Mahzar Fareed and Claudio Zandron has been partially supported by the Università degli Studi di Milano-Bicocca, "Fondo Ateneo per la Ricerca, quota competitiva", project 22022-ATEQC-0046.

References

1. Baquero, F., Campos, M., Llorens, C.G., Sempere, J.M.: P systems in the time of covid-19. Journal of Membrane Computing **3** (2021) 246–257
2. Anderson, R.M., May, R.M.: Infectious diseases of humans: dynamics and control. Oxford university press (1991)
3. Manfredi, P., D'Onofrio, A.: Modeling the interplay between human behavior and the spread of infectious diseases. Springer Science & Business Media (2013)
4. Wang, Z., et al.: Statistical physics of vaccination. Physics Reports **664** (2016) 1–113
5. Paun, G.: Membrane computing: an introduction. Springer Science & Business Media (2012)
6. Sheikh, A., et al.: Sars-cov-2 delta voc in scotland: demographics, risk of hospital admission, and vaccine effectiveness. Lancet **397**(10293) (2021) 2461–2462
7. Bullard, J., et al.: Predicting infectious severe acute respiratory syndrome coronavirus 2 from diagnostic samples. Clinical Infectious Diseases **71**(10) (2020) 2663–2666

8. Romano, A.: Chapter I dati delle piattaforme durante la crisi da COVID-19 (2022)
9. Cerqueira-Silva, T., et al.: Influence of age on the effectiveness and duration of protection of vaxzevria and coronavac vaccines: A population-based study. Lancet Regional Health - Americas **6** (2022) 100–101

Neurons on wifi

David Orellana-Martín[1,2], Francis George C. Cabarle[1,2,3], Prithwineel Paul[4], XiangXiang Zeng[5], Rudolf Freund[6]

[1]Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[2]SCORE lab, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: dorellana@us.es, fcabarle@us.es
[3]Department of Computer Science, University of the Philippines Diliman,
1101 Quezon City, Philippines
E-mail: fccabarle@up.edu.ph
[4]Department of Computer Science and Engineering, Institute of Engineering and Management, University of Engineering and Management, New Town Rd., Kolkata, 700091, India
E-mail: prithwineel.paul@iem.edu.in
[5]Deparment of Computer Science, Hunan University, Changsha, China
E-mail: xzeng@hnu.edu.cn
[6]Faculty of Informatics, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria
E-mail: rudi@emcc.at

Summary. Spiking neural P systems, SN P systems in short, are membrane systems based on the third generation of neuron models (spiking neurons). Recent results in neuroscience highlight the importance of *extrasynaptic* activities of neurons, that is, features and functioning of neurons apart from their synapses. Previously it was thought that signals such as *neuropeptides* only assist neurons but such signals are given further importance more recently. Inspired by such recent results, we introduce the idea of *wireless SN P systems*, or *WSN P systems* in short. In WSN P systems no synapses exist, and we associate regular expressions for each neuron to decide which spikes it receives. We provide two semantics of how to "interpret" the spikes released by neurons. A specific register machine is simulated to show how different the programming style is with WSN P systems compared to SN P systems and other variants. The programming style emphasises a trade-off: WSN P systems can be more "flexible" in the sense that neurons are not limited by their synapses as before for sending spikes; the loss of the useful and directed graph, however, requires careful design of the rules and the regular expression associated with each neuron. For instance, in the present work we make use of prime numbers to create the expressions and rules of the neurons.

Keywords: Membrane computing, spiking neural P systems, extrasynaptic signalling, neuropeptides

1 Introduction

The present work introduces a variant of spiking neural P systems, in short SN P systems. SN P systems introduced in [1] are inspired by spiking neurons and their network: the processors are neurons which are the nodes in a directed graph; the edges are called synapses, which allow the communication between neurons of a single object a referred to as a spike; the neurons are spike processors which consume and produce spikes.

Some recent survey papers of SN P systems and variants include [2, 3] and more recently in [4]. Since their introduction, it is known that SN P systems are Turing complete. SN P systems can also solve **NP**-complete problems, trading time for space [5]. In the past decade or so many variants of SN P systems have been introduced depending on specific ingredients or features, mostly from biology. For instance the introduction of autapses [6], synaptic plasticity [7], synaptic schedules [8], neurogenesis [9].

Besides theoretical works, simulators of SN P systems and variants are used to support research or pedagogy, such as interactive and visual software in [10, 11] with the main page in [12], and recent tutorial in [13]. Solutions to hard problems are also implemented in parallel hardware such as in [14] which implements ideas from [15], with recent and some state-of-the-art results in [16].

In the present work we introduce the idea of wireless SN P systems, or WSN P systems in short. One general reference for the bio-inspiration of WSN P systems is from [17] with recent and detailed results from [18] and [19]. Briefly, such recent results emphasise the crucial and important role of neuronal activities outside of their synapses, hence their *wireless* features and functions. Such recent works focus their attention on a specific animal known as *C. elegans*.

The worm *C. elegans* is a model organism, that is, much is known about its biology including its nervous system due to its "simplicity" of several hundred neurons only. Despite the small size of this worm, its nervous system has interesting biochemical complexity with structural features shared by larger animals [19]. Due to better techniques and technology, more recently there are improved works to show how a *wireless network* (that is, without synaptic wiring) among nerve cells or neurons is able to operate [18, 19]. These recent works challenge the idea neurons communicate only or mainly through anatomical connections, that is, through their synapses [17]. Such recent works reveal new details of a *connectome* or wiring diagram among neurons, the *neuropeptidergic connectome*: a connectome which is equally important and perhaps more diverse than the synaptic connectome.

Furthermore, these recent works identify *neuropeptides*, the chemical messages released by neurons, as the basis for such wireless network among neurons. Neurons in the *C. elegans* worms can release neuropeptides, or have receptors for such neuropeptides. The wireless network formed from these *pairs of releasing* and *receiving neurons* is dense and decentralised, compared to the less dense and more centralised network of synapses [19]. Such pairs are responsible for existence of the wireless network, which means that neuropeptides are not simply random chemicals floating between neurons. Neuropeptides affect the neural system over larger scales of time and space, unlike synaptic signals restricted only to both sides of the synapse [19]

Previously it was thought that neuropeptides only assisted in synaptic communication. However, these recent works indicate the ubiquitous, important, and *direct role to neuron activation* of neuropeptides and the corresponding wireless network [17]. Neuropeptides are conserved and ancient chemicals in brains of many organisms, including humans brains, suggesting the pioneering work with *C. elegans* can at least reveal useful structures or principles for brain function [18, 19]. For instance, a recent technique allows to detect neuropeptides, which can assist in better understanding of both wired and wirless networks of neurons including those for humans [20].

We use the recent results mentioned as inspirations for *extrasynaptic* functions of neurons, that is, functioning without or outside the usual synapses. The contribution of the present work is the introduction of wireless SN P systems. No synapses are present in the neurons, while still using rules to consume and produce spikes. For each neuron we associate a regular expression to decide what "forms" of spikes the neuron can receive. We introduce two semantics for WSN P systems, based on the interpretation of the spikes released at each step by the neurons: the spike package semantic considers the spikes as individual packages as released by each neuron; the spike total semantic considers the sum of spikes released by all neurons. We show how to programme a specific WSN P system through the simulation of a specific register machine. Such a simulation emphasises the rather different way to programme WSN P systems compared SN P systems and variants, due to the associated expression for each neuron and the lack of synapses. In this way we note that the directed graph structure of SN P systems and variants is a very useful feature. Some "flexibility" is gained in the sense that the neurons are not limited to sending spikes only to neurons where their synapses connect. However, losing the directed graph makes the programming of the system more "involved" in the sense that more effort can be required to design the rules of each neuron.

The present work is organised as follows: in the next Section 2 we provide in an intuitive way an example of a WSN P system Π_1. We examine the computations of Π_1 under two semantics, the spike package and spike total semantics. In Section 3 we show how a WSN P system can simulate a small and specific register machine to highlight the rather different way to programme such systems. Lastly, in Section 4 we provide some conclusions and directions for further work.

2 An example with two semantics

In this section we consider an example, the system Π_1 shown in Figure 1. We use Π_1 to elaborate two semantics about wireless SN P systems. Briefly, Π_1 has 3 neurons, each labelled with a pair (i, E_i) for $1 \leq i \leq 3$. Each neuron has an associated regular expression to check what number of spikes it can receive. For instance, neurons σ_1 and σ_2 have $E_1 = E_2 = a$ which means they only receive spikes of the form $a^1 = a$ fired from other neurons, including from σ_1 itself. We note that the rule set of σ_2 is empty, so later we see the number of spikes inside it either remain the same or increase.

We omit the definition, syntax, and semantics standard to SN P systems. The reader is referred instead for instance to the seminal paper [1], in open access tutorials or surveys as in [21, 2], or the dedicated chapter of the handbook in [22].

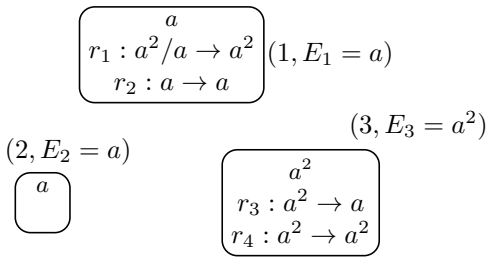


Fig. 1: Π_1 is an example of a wireless SN P system.

2.1 Semantic 1: spike package

The semantic 1 we first consider, which we refer to as *spike package semantic,* considers only in spikes arriving in "packages" sent by neurons in the environment. Consider two neurons which fire at the same step t: let neuron σ_i and σ_j have regular expressions $E_i = a^m$ and $E_j = a^n$ associated, respectively, for $n, m \geq 1$; neuron σ_i and σ_j fire a^n and a^m spikes at step t, respectively. At the next step $t + 1$, neuron σ_i receives the a^m spikes from neuron σ_j, and vice-versa. That is, while at step t there is a total of $n + m$ spikes in the environment due to the firing of both neurons: in spike package semantic we only consider packages or groups of the spikes and not the total spikes in the environment. We consider the spike total semantic as semantic 2 in Section 2.2 later.

Let us now apply the spike package semantic to the system Π_1 in Figure 1. To help with clarifying the computation of Π_1 we refer to the configuration tree in Figure 2 under spike package semantic.

The initial configuration of Π_1, according to the total ordering of 1, 2, and 3 of the neurons, is $C_0 = \langle 1, 1, 2\rangle$. That is neurons 1, 2, and 3 each have 1, 1, and 2 spikes, respectively. Due to C_0 and the nondeterminism in Π_1 found only in neuron σ_3, there is a choice between applying rule r_2, and either r_3 or r_4.

If rule r_2 is applied one spike is consumed in neuron σ_1, and sent to both σ_1 and neuron σ_2 due to their associated regular expressions $E_1 = E_2 = a$. Applying r_3 means σ_3 consumes two spikes but produces only one spike. Again the single spike from σ_3 arrives at σ_1 and σ_2 due to their regular expressions. Hence, we have the transition $C_0 \overset{r_2 r_3}{\Longrightarrow} C_{1,0} = \langle 2, 3, 0\rangle$, that is, by applying r_2 and r_3 we obtain configuration $C_{1,0}$ from C_0.

Consider now if we apply r_2 and r_4 instead. The effect applying of r_2 is still to return a spike to σ_1 and to increase the spikes in σ_2. The effect of r_4 is *reflexive*, that is, in neuron σ_3 two spikes are consumed and then returned to itself since $E_3 = a^2$. Hence, we have the transition $C_0 \overset{r_2 r_4}{\Longrightarrow} C_{1,1} = \langle 1, 2, 2\rangle$, that is, by applying r_2 and r_4 we obtain configuration $C_{1,1}$ from C_0.

As seen in the configuration tree in Figure 2, each branch of computation of Π_1 is nonhalting, that is, Π_1 always arrives at a configuration where some rule is applied. The number of spikes in neuron σ_2 continue to increase. More precisely, we have transition $\langle 2, b, 0\rangle \overset{r_1}{\Longrightarrow} \langle 1, b, 2\rangle$, transition $\langle 1, b, 2\rangle \overset{r_2 r_3}{\Longrightarrow} \langle 2, b + 2, 0\rangle$, or transition $\langle 1, b, 2\rangle \overset{r_2 r_3}{\Longrightarrow} \langle 1, b + 1, 2\rangle$ for some $b \geq 1$.

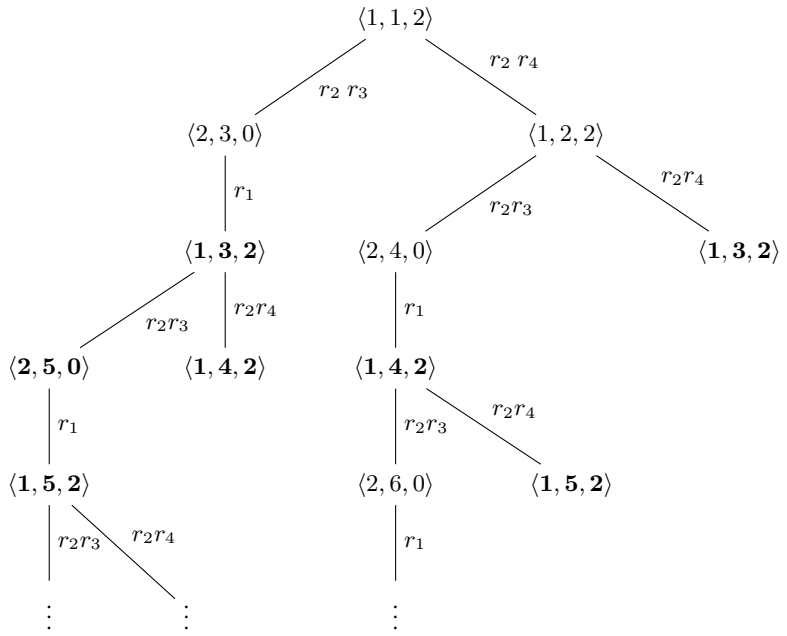


Fig. 2: A tree of configurations of Π_1 in Figure 1 using semantic 1 (spike package semantic). The initial configuration is $\langle 1, 1, 2\rangle$. Except for $\langle 1, 1, 2\rangle$, each *node* in the tree is a next configuration by applying the rules labelling the connecting *edge*. Nodes or configurations in bold are nodes repeated elsewhere in the portion of the tree shown.

2.2 Semantic 2: spike total

We continue the same notation at the start of Section 2.1 to consider the total spike semantic. Recall we have neurons with labels and their associated expressions as $\sigma_i = (i, E_i = a^m)$ and $\sigma_j = (j, E_j = a^n)$ for $n, m \geq 1$. At step t neurons σ_i and σ_j fire n and m spikes, respectively. Thus we have a total of $n + m$ spikes in the environment. In the next step $t + 1$, no neuron receives any spikes since $a^{n+m} \notin L(E_i)$ and $a^{n+m} \notin L(E_j)$. That is, none of the regular expressions of both neurons describe the total number of spikes in the environment.

Consider now the same SN P system Π_1 from Figure 1 but under the total spikes semantic. The configuration tree of Π_1 is now given by Figure 3. From the same initial configuration $C_0 = \langle 1, 1, 2\rangle$ the computation proceeds in a different way. The transition $C_0 \overset{r_2 r_4}{\Longrightarrow} C_{1,1} = \langle 0, 1, 0\rangle$ is a halting configuration, that is, no more rules can be applied in Π_1. Only the subtree with transition $C_0 \overset{r_2 r_3}{\Longrightarrow} C_{1,0} = \langle 0, 1, 2\rangle$ continues to infinitely grow the number of spikes in neuron σ_2. Actually after configuration $C_{2,0} = \langle 1, 2, 0\rangle$ only rule r_2 can be applied in a nonhalting computation.

We note that the effect of applying rules r_2 and r_4 from C_0 is to release a total of a^3 spikes in the environment followed by the halting of Π_1. Since we use the total spikes

semantic no neuron receives these spikes in the next step because no neuron has an expression which includes a^3.

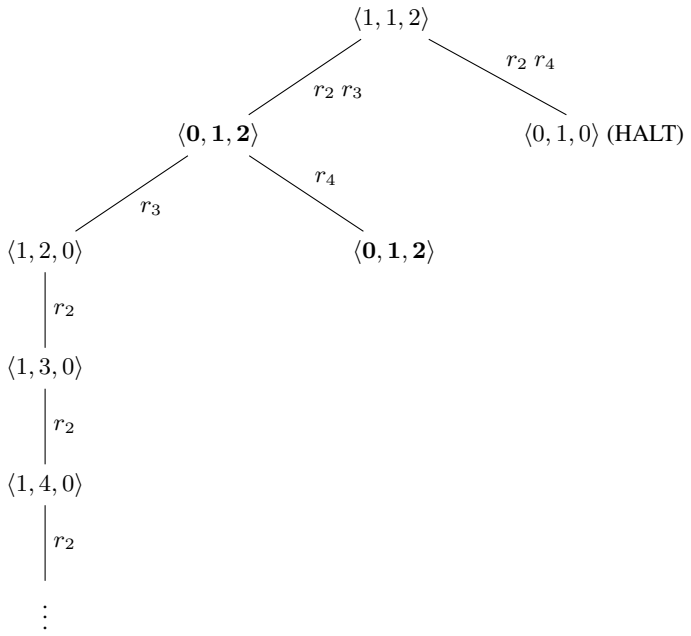


Fig. 3: Configuration tree for Π_1 in Figure 1 using semantic 2 (total spikes semantic). As in Figure 2, edges between nodes (configurations) are labelled by the rules applied from the source to destination nodes. Also, configurations in bold means they are repeated elsewhere in the tree.

3 Programming WSN P systems

Let us consider a small programme with some register machine M to give us an idea how to programme a WSN P system, including their similarities and differences with SN P systems and their other variants. It is known that register machines compute the set of all Turing computable sets of numbers [23]. We do not go into the details of register machines here, and refer the reader instead to [23] as well as to [1, 22] for the usual style of proofs with register machines. Consider the following instructions of a register machine M :

$l_1 : (SUB(r_1), l_2, l_3)$,
$l_2 : (ADD(r_1), l_1, l_3)$,
$l_3 : HALT$.

We simulate the instructions of M using a WSN P system Π_M with the following details. We map prime numbers to elements of M and use the mapping as *addresses* in Π_M. The

idea of addresses and simulation is made clear in a moment. In general, for elements of any register machine we use the total order $l_1, l_2, \ldots, r_1, r_2, \ldots$. Specific to M we have the following mapping of its elements to prime numbers:

$$p_{l_1} = 7, p_{l_2} = 11, p_{l_3} = 13, p_{r_1} = 17.$$

That is, starting from instruction l_1 of M we map it to the prime number $p_{l_1} = 7$, followed by mapping l_2 and l_3 to $p_{l_2} = 11$ and $p_{l_3} = 13$, respectively. After mapping prime numbers to all instructions of M, we map the next prime numbers to registers: there is only one register in M mapped to $p_{r_1} = 17$.

In general, the mapping we use for the content of register $r_i = n$ is having $a^{2p_{r_i}n}$ spikes in the neuron σ_{r_i}. Following the mapping of prime numbers above to elements of M: if $r_1 = n$ the associated neuron σ_{r_1} has $a^{2p_{r_1}n} = a^{2(17)n}$ spikes.

3.1 Simulating a SUB instruction

Now we provide the SUB module of Π_M to simulate instruction l_1 of M. The SUB module consists of the following neurons and their contents. We note that the contents of a neuron $\sigma_i = (a^n, R_i, E_i)$ consists of its initial number of n spikes, its rule set R_i, and the associated regular expression E_i.

$\sigma_{l_1} = (a^7, R_{l_1}, E_{l_1} = a^7)$,
$\sigma_{aux_{1,1}} = (\lambda, R_{aux_{1,1}}, E_{aux_{1,1}} = a^{2(17)}(a^{17})^+)$,
$\sigma_{r_1} = (a^{2(17)n}, R_{r_1}, E_{r_1} = a^{17} \cup a^{2(17)})$.

The rule sets of each neuron we list as follows.

$R_{l_1} = \{a^7 \to a^{7(17)}\}$,
$R_{aux_{1,1}} = \{a^{7(17)}/a^{7(17-1)} \to a^{17}, a^{7+3(17)} \to a^{11}, a^{7+5(17)} \to a^{13}\}$,
$R_{r_1} = \{(a^{2(17)})^+ a^{17}/a^{3(17)} \to a^{3(17)}, a^{17} \to a^{5(17)}\}$.

The rules in each rule set are written in an explicit way with their superscripts, to make it easier to see the idea of the simulation. For instance, in simulating instruction l_1, neuron σ_{l_1} has only one rule releasing $a^{p_{l_1}(p_{r_1})} = a^{7(17)}$ spikes to mean the following: the source of spikes is σ_{l_1} with σ_{r_1} as destination. To simulate the next instruction, neuron $\sigma_{aux_{1,1}}$ releasing either $p_{l_2} = 11$ or $p_{l_3} = 13$ spikes means the destination neuron is either σ_{l_2} or σ_{l_3}, respectively.

Now we simulate instruction l_1 of M by the SUB module of Π_M as follows. Consider a total order of neurons in the SUB module of Π_M as $\sigma_{l_1}, \sigma_{aux_{1,1}}, \sigma_{r_1}$. From the above description, the initial configuration at time step $t = 0$ of the total order is given by $C_0 = \langle 7, 0, 2(17)n \rangle$.

At step $t = 1$, the a^7 spikes in neuron σ_{l_1} start the computation by applying the single rule in the neuron: all $p_{l_1} = 7$ spikes are consumed and $7(17) = p_{l_1}(p_{r_1})$ spikes are produced. The reason for $7(17)$ spikes is to indicate that instruction l_1 sends its spikes to perform subtraction operation on register r_1. At step 1 have the configuration $C_1 = \langle 0, 7(17), 2(17)n \rangle$. When the spikes have been sent, only the auxiliary neuron $\sigma_{aux_{1,1}}$ receives the spikes from σ_{l_1} since only the regular expression associated with $\sigma_{aux_{1,1}}$ makes a match. That is, we have $a^{7(17)} \in L(E_{aux_{1,1}})$.

At step $t = 2$, only the rule $a^{7(17)}/a^{7(17-1)} \to a^{17}$ of $\sigma_{aux_{1,1}}$ is applied: it consumes $7(17-1)$ spikes and produces 17 spikes is received only by neuron σ_{r_1}. At step 2 we have the configuration $C_2 = \langle 0, 7, 2(17)n + 17\rangle$.

At step $t = 3$ only neuron σ_{r_1} can apply a rule. Depending on the value of n in register r_1 of M, we have the following two cases:

1. if $n > 0$, this means that before $t = 2$, neuron σ_{r_1} has $2(p_{r_1})n = 2(17)n \geq 34$ spikes. Let $n = 1$. Then, receiving 17 spikes means the total spikes in σ_{r_1} at step $t = 2$ is $17 + 34n = 51$ spikes.
   The first rule of σ_{r_1} is applied since $a^{51} \in L((a^{2(17)})^+ a^{17})$. Applying the rule consumes $3(17)$ spikes, no spikes remain in σ_{r_1} since $51 - 3(17) = 0$. In this way, as the number in register r_1 is reduced from n to $n-1$, the number of spikes in σ_{r_1} is reduced from $a^{2(17)n}$ to $a^{2(17)(n-1)}$. The rule also produces $a^{3(17)}$ spikes which in step $t = 4$ only $\sigma_{r_{1,1}}$ receives.
   At the moment $t = 4$ the configuration is $C_4 = \langle 0, 7+3(17), 2(17)(n-1)\rangle$ and only $\sigma_{r_{1,1}}$ can apply a rule: the neuron applies the rule $a^{7+3(17)} \to a^{11}$ to consume all of its spikes and to activate the next module to simulate instruction l_2 associated with $p_{l_2} = 11$.
2. if $n = 0$, before step $t = 2$ neuron σ_{r_1} has no spikes. Receiving 17 spikes means the total spikes in σ_{r_1} at moment $t = 3$ is 17 spikes: the neuron applies its rule $a^{17} \to a^{5(17)}$ to consume all spikes and send spikes only to $\sigma_{r_{1,1}}$. In this way, as the number in register r_1 is 0, the number of spikes in σ_{r_1} remains 0 also.
   At the moment $t = 4$ the configuration is now $C_4 = \langle 0, 7+5(17), 0\rangle$, with only $\sigma_{r_{1,1}}$ applying a rule: the rule $a^{7+5(17)} \to a^{13}$ is applied, consuming all spikes. At the next step the simulation of instruction l_3 associated with $p_{l_3} = 13$ begins.

Thus, the subtraction instruction l_1 of M is correctly simulated: if register r_1 contains a nonzero value it is decremented and the next instruction is l_2, otherwise r_1 remains zero and l_3 is the next instruction. We note that there is no interference in the case when there is more than one subtraction instruction associated with r_1. The mapping of prime numbers over a total ordering on M described above, and the "addresses" of each neuron based on the mapping allows no wrong simulation. Such addresses we use not only in the regular expressions associated with each neuron, but also in the spikes released by each neuron.

3.2 Simulating an ADD instruction

This section is devoted to the ADD module of Π_M to simulate instruction l_2 of M provided at the start of Section 3. The ADD module consists of the following neurons and their contents. For simulating this, we must include new rules in $R_{aux_{1,1}}$

$\sigma_{l_2} = (\lambda, R_{l_2}, E_{l_2} = a^{11})$,

The rule sets of each neuron we list as follows.

$R_{l_2} = \{a^{11} \to a^{11(17)}\}$,

$R_{aux_{1,1}} = R_{aux_{1,1}} \cup \{a^{11(17)}/a^{11(17-1)} \to a^{2(17)}, a^{11} \to a^7, a^{11} \to a^{13}\}$

Let us suppose that, at step $t = k$, a^{11} spikes arrive to neuron σ_{l_2}. Then, the simulation of the instruction l_2 starts. Consider a total order of neurons in the ADD module of Π_M as

$\sigma_{l_2}, \sigma_{aux_{1,1}}, \sigma_{r_1}$. From the above description, the configuration at the time $t = k$ neuron σ_{l_2} receives a^{11} spikes of the total order is given by $C_k = \langle 11, 0, 2(17)n \rangle$.

At step $t = k + 1$ the a^{11} spikes in neuron σ_{l_2} start the simulation by applying the single rule in the neuron: all $p_{l_2} = 11$ spikes are consumed and $11(17) = p_{l_1}(p_{r_1})$ spikes are produced. Similar to the SUB instruction, the reason for $11(17)$ spikes is to indicate that instruction l_2 sends its spikes to perform addition to register r_1. When the spikes have been sent, only the auxiliary neuron $\sigma_{aux_{1,1}}$ receives the spikes from σ_{l_1} since only the regular expression associated with $\sigma_{aux_{1,1}}$ makes a match. That is, we have $a^{11(17)} \in L(E_{aux_{1,1}})$.

At step $t = k + 2$, only the rule $a^{11} \to a^{11(17)}$ of $\sigma_{aux_{1,1}}$ is applied: it consumes $11(17 - 1)$ spikes and produces $2(17)$ spikes. Only neuron σ_{r_1} will receive the spikes. Thus, $C_{k+2} = \langle 0, 11, 2(17)(n + 1) \rangle$.

At step $t = k + 3$, both rules $a^{11} \to a^7$ and $a^{11} \to a^{13}$ are applicable, so one of them is selected in a non-deterministic way. If the first one is applied, a^7 spikes are be fired, matching with the regular expression of neuron σ_{l_1}. Otherwise, rule a^{13} spikes are sent to neuron σ_{l_3} mapped to $p_{l_3} = 13$. In the first case, the SUB instruction is simulated again, while in the second case the output must be produced followed by the halting of Π_M.

Thus, the addition instruction l_2 of M is correctly simulated: the value of register r_1 is augmented by 1 and the next instruction is selected from the set $\{l_1, l_3\}$ in a non-deterministic way. No interference with rules from the SUB instruction is found. The regular expressions always match with the prime number p_{l_2} corresponding with instruction l_2. That is, p_{l_2} spikes are never released while simulating the SUB instruction.

Before we end the present section on programming Π_M to simulate M we make a few more notes. First we omit the explicit simulation of instruction l_3 to halt M. In Π_M, simulating a halt instruction requires the release of the output to the environment. In the above description of Π_M we assume the use of spike package semantic as in Section 2.1. It seems to be the case that Π_M as described above can still simulate M under the spike total semantic in Section 2.2.

4 Final remarks

We introduced yet another variant of SN P systems we refer to as wireless SN P systems, or WSN P systems in short. Several kinds of novelty can be found in WSN P systems. The further movement away from a fixed or static graph motivated especially by recent and exciting discoveries in neuroscience. That is, our increasing knowledge of extrasynaptic signalling, of neuropeptides and their important influence in neuronal activities. WSN P systems go against the traditional directed graph used in neural systems or networks, introducing two semantics how the "floating" spikes are received. We associate regular expressions to each neuron allowing neurons to distinguish which spikes to accept or reject. Both semantics, the spike partial and spike total, are bio-inspired. The semantics also bear some resemblance to packets of data among networks of computers that connect for instance wireless networks and the Internet.

The use of forgetting rules of the form $a^s \to \lambda$, are common to SN P systems and many variants. Forgetting rules are used to remove spikes without producing spikes, but

such rules may not be necessary in WSN P systems. A way to avoid such rules is to use a rule $a^s \to a^x$ where x is not found in any regular expression of any neuron. In this way we still remove the s number of spikes, but more care needs to be given using the spike total semantic (Section 2.2). The output neuron mentioned at the end of Section 3.2 needs to be a distinguished neuron, in order to obtain the output of the system.

In many variants of SN P systems the delay feature is common: there can be a nonzero delay from releasing a spike and the spike arriving to another neuron. It is known, see e.g. [24], that the delay feature is not required for universality, but can be useful for instance in modelling [25]. It is interesting to see the role of delays in WSN P systems. Other common features of SN P systems and variants include the lack of reflexive synapses, and restricting the produced spikes of a neuron to be at most the consume spikes. A variant known as SN P systems with autapses allows reflexive synapses although this variant has a static and directed graph [6]. For restricting the produced spikes to be less or equal to the consumed spikes, perhaps this can be achieved by using more time in the computation, and more neurons to generate the required spikes.

Regarding the semantics in Section 2, it is interesting to see which types of problems or computations one semantics has an advantage over the other. As seen in the configuration trees in Figure 2 and Figure 3, for the same Π_1 the computations are rather different. Another interesting extension or semantic for WSN P systems is the idea of decay or "attenuation" of spikes: it is assumed that spikes (especially if delays are introduced) can "float" without change for an arbitrary duration in time or distance in space between neurons. It is interesting to introduce such decay or attenuation in WSN P systems, similar to decay of electromagnetic signals used in wireless networks of computers. Previously, decaying spikes were considered in SN P systems [26].

In programming Π_M we notice its operation is sequential, that is, at each step at most one neuron applies a rule. The sequential restriction or normal form has been applied to SN P systems as early as in [27], and more recently with variants having dynamic topologies in [28, 29]. It is interesting what kinds of restrictions and computations can(not) be obtained when more parallelism is involved in the system in terms of neurons, rules, etc.

Another interesting direction is to consider matrix representations of WSN P systems, as done with SN P systems in [30] and more recently in [31, 11]. Such representations allow for faster simulations, such as parallel processors [16, 32] web browsers [10, 11, 33], and their automatic design [34, 35]. The related variant with matrix representation seems to be SNPSP systems in [36]. SNPSP systems introduce plasticity to allow adding or removing of synapses, introduced in [7]. Another variant known as SNP systems with scheduled synapses (in short, SSNP systems) has synapse dynamism, by assigning schedules or (range of) time steps when synapses exist or not. Besides SNPSP systems and SSNP systems, another related variant are extended SN P systems in [37] which also have no fixed and directed graph structure. It is also interesting to consider WSN P systems in the formal framework of [38] for membrane systems and related models.

A few other lines of investigation on computing power to consider are the following. Computing languages with WSN P systems, for instance in [39, 40]. Providing "small" WSN P systems as in [41, 42]. Normal forms such as restricting the types of regular expressions, with optimal results in [24]. In the case of WSN P systems not only are there

expressions in rules but also those associated with the neurons. Creating homogeneous systems as in [43, 44] is also worth investigating: the expressions associated for some neurons must be distinct but not for others; also the number of produced spikes may need to be heterogeneous for some rules, unlike previous works on homogeneous SN P systems where each neuron has the same rule set.

Besides computing power, computing efficiency is interesting to consider with WSN P systems. For instance how to solve **NP**-complete problems in a (non-)uniform way [45, 5]. An interesting extension is the feature to allow creation of new neurons as in [9, 46] or using the idea of pre-computed resources [47]. Real world applications can perhaps benefit from WSN P systems with neurons having the ability to "distinguish signals" using their associated expressions. Applications may include improvements on intrusion detection [48] and skeletonizing images [49]. More directions and open problems can be derived from [2, 4].

We end the present work by highlighting, based on the ideas here presented, that the directed graph structure of an SN P system seem to be powerful, at least useful, in programming the system. Losing such directed graph as shown in WSN P systems we need to use regular expressions for each neuron. Besides, here we use a mapping of prime numbers for simulating a register machine: a rather unconventional way of simulation at least in terms of the usual way of simulating register machines with such membrane systems. These ideas show that the programming of WSN P systems are quite different and interesting compared to SN P systems and their many variants.

Acknowledgements

F.G.C. Cabarle is supported by the QUAL21 008 USE project, "Plan Andaluz de Investigación, Desarrollo e Innovación" (PAIDI) 2020 and "Fondo Europeo de Desarrollo Regional" (FEDER) of the European Union, 2014-2020 funds. X. Zeng was supported by National Natural Science Foundation of China (Grant Nos. 62122025, U22A2037, 62250028)

References

1. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural p systems. Fundamenta informaticae **71**(2, 3) (2006) 279–308
2. Leporati, A., Mauri, G., Zandron, C.: Spiking neural p systems: main ideas and results. Natural Computing **21**(4) (2022) 629–649
3. Fan, S., Paul, P., Wu, T., Rong, H., Zhang, G.: On applications of spiking neural p systems. Applied Sciences **10**(20) (2020) 7011
4. Cabarle, F.G.C.: Thinking about spiking neural p systems: some theories, tools, and research topics. Journal of Membrane Computing (2024) 1–20
5. Leporati, A., Mauri, G., Zandron, C., Păun, G., Pérez-Jiménez, M.J.: Uniform solutions to sat and subset sum by spiking neural p systems. Natural computing **8**(4) (2009) 681–702
6. Song, X., Valencia-Cabrera, L., Peng, H., Wang, J.: Spiking neural p systems with autapses. Information Sciences **570** (2021) 383–402

7. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking neural p systems with structural plasticity. Neural Computing and Applications **26** (2015) 1905–1917
8. Cabarle, F.G.C., Adorna, H.N., Jiang, M., Zeng, X.: Spiking neural p systems with scheduled synapses. IEEE Transactions on Nanobioscience **16**(8) (2017) 792–801
9. Wang, J., Hoogeboom, H.J., Pan, L.: Spiking neural p systems with neuron division. In: Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers 11, Springer (2011) 361–376
10. Gulapa, M., Luzada, J.S., Cabarle, F.G.C., Adorna, H.N., Buño, K., Ko, D.: Websnapse reloaded: The next-generation spiking neural p system visual simulator using client-server architecture. In: Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023), Atlantis Press (2024) 434–461
11. Gallos, L., Sotto, J.L., Cabarle, F.G.C., Adorna, H.N.: Websnapse v3: Optimization of the web-based simulator of spiking neural p system using matrix representation, webassembly and other tools. In: Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023), Atlantis Press (2024) 415–433
12. : Websnapse page (2023) `https://aclab.dcs.upd.edu.ph/productions/software/websnapse`.
13. Ko, D., Cabarle, F.G.C., De L Cruz, R.T.: WebSnapse Tutorial: A Hands-On Approach for Web and Visual Simulations of Spiking Neural P Systems. In: Bulletin of the International Membrane Computing Society. Volume 16. (December 2023) 137–153
14. Hernández-Tello, J., Martínez-Del-Amor, M.A., Orellana-Martín, D., Cabarle, F.G.: Sparse matrix representation of spiking neural p systems on gpus. In: International Conference on Membrane Computing, Chengdu, China and Debrecen, Hungary (August 2021) 316–322
15. Martínez-Del-Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural p systems with sparse matrix-vector operations. Processes **9**(4) (2021)
16. Hernández-Tello, J., Martínez-Del-Amor, M.Á., Orellana-Martín, D., Cabarle, F.G.C.: Sparse spiking neural-like membrane systems on graphics processing units. International Journal of Neural Systems **34**(7) (2024) 2450038–2450038
17. Lloreda, C.L.: Wi-fi for neurons: first map of wireless nerve signals unveiled in worms. Nature **623**(7989) (2023) 894–895
18. Randi, F., Sharma, A.K., Dvali, S., Leifer, A.M.: Neural signal propagation atlas of caenorhabditis elegans. Nature (2023) 1–9
19. Ripoll-Sánchez, L., Watteyne, J., Sun, H., Fernandez, R., Taylor, S.R., Weinreb, A., Bentley, B.L., Hammarlund, M., Miller, D.M., Hobert, O., Beets, I., Vértes, P.E., Schafer, W.R.: The neuropeptidergic connectome of c. elegans. Neuron **111**(22) (2023) 3570–3589.e5
20. Wang, H., Qian, T., Zhao, Y., Zhuo, Y., Wu, C., Osakada, T., Chen, P., Chen, Z., Ren, H., Yan, Y., Geng, L., Fu, S., Mei, L., Li, G., Wu, L., Jiang, Y., Qian, W., Zhang, L., Peng, W., Xu, M., Hu, J., Jiang, M., Chen, L., Tang, C., Zhu, Y., Lin, D., Zhou, J.N., Li, Y.: A tool kit of highly selective and sensitive genetically encoded neuropeptide sensors. Science **382**(6672) (2023) eabq8173
21. Gh. Păun: Spiking neural P systems: A tutorial. Bull. Eur. Assoc. Theor. Comput. Sci. **91** (Feb 2007) 145–159
22. Gh. Păun, Rozenberg, G., Salomaa, A., eds.: The Oxford Handbook of Membrane Computing. Oxford Univeristy Press (2010)
23. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., USA (1967)
24. Macababayao, I.C.H., Cabarle, F.G.C., de la Cruz, R.T.A., Zeng, X.: Normal forms for spiking neural p systems and some of its variants. Information Sciences **595** (2022) 344–363

25. Cabarle, F.G.C., Buño, K.C., Adorna, H.N.: Time after time: notes on delays in spiking neural p systems. In: Theory and Practice of Computation: 2nd Workshop on Computation: Theory and Practice, Manila, The Philippines, September 2012, Proceedings, Springer (2013) 82–92
26. Freund, R., Ionescu, M., Oswald, M.: Extended spiking neural p systems with decaying spikes and/or total spiking. International Journal of Foundations of Computer Science **19**(05) (2008) 1223–1234
27. Ibarra, O.H., Woodworth, S., Yu, F., Păun, A.: On spiking neural p systems and partially blind counter machines. In: Unconventional Computation: 5th International Conference, UC 2006, York, UK, September 4-8, 2006. Proceedings 5, Springer (2006) 113–129
28. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J.: Sequential spiking neural p systems with structural plasticity based on max/min spike number. Neural computing and applications **27** (2016) 1337–1347
29. Bibi, A., Xu, F., Adorna, H.N., Cabarle, F.G.C., et al.: Sequential spiking neural p systems with local scheduled synapses without delay. Complexity **2019** (2019)
30. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural p systems. In: Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers 11, Springer (2011) 377–391
31. Adorna, H.N.: Matrix representations of spiking neural p systems: Revisited. arXiv preprint arXiv:2211.15156 (2022)
32. Aboy, B.C.D., Bariring, E.J.A., Carandang, J.P., Cabarle, F.G.C., De La Cruz, R.T., Adorna, H.N., Martínez-del Amor, M.Á.: Optimizations in cusnp simulator for spiking neural p systems on cuda gpus. In: 2019 International Conference on High Performance Computing & Simulation (HPCS), IEEE (2019) 535–542
33. Odasco, A.N.L., Rey, M.L.M., Cabarle, F.G.C.: Improving gpu web simulations of spiking neural p systems. Journal of Membrane Computing (2023) 1–16
34. Gungon, R.V., Hernandez, K.K.M., Cabarle, F.G.C., De la Cruz, R.T.A., Adorna, H.N., Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I.: Gpu implementation of evolving spiking neural p systems. Neurocomputing **503** (2022) 140–161
35. Uy, A.V.D., Wu, J.J.Q.C., Cabarle, F.G.C., de la Cruz, R.T.A., Adorna, H.N.: Evolving spiking neural p systems with rules on synapses on cuda. Philippine Computing Journal **17** (2022) 10–31
36. Jimenez, Z.B., Cabarle, F.G.C., de la Cruz, R.T.A., Buño, K.C., Adorna, H.N., Hernandez, N.H.S., Zeng, X.: Matrix representation and simulation algorithm of spiking neural p systems with structural plasticity. Journal of Membrane Computing **1** (2019) 145–160
37. Alhazov, A., Freund, R., Oswald, M., Slavkovik, M.: Extended spiking neural p systems. In Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A., eds.: Membrane Computing, Berlin, Heidelberg, Springer Berlin Heidelberg (2006) 123–134
38. Verlan, S., Freund, R., Alhazov, A., Ivanov, S., Pan, L.: A formal framework for spiking neural p systems. Journal of Membrane Computing **2**(4) (2020) 355–368
39. Chen, H., Freund, R., Ionescu, M., Păun, G., Pérez-Jiménez, M.J.: On string languages generated by spiking neural p systems. Fundamenta Informaticae **75**(1-4) (2007) 141–162
40. Cabarle, F.G.C., de la Cruz, R.T.A., Zhang, X., Jiang, M., Liu, X., Zeng, X.: On string languages generated by spiking neural p systems with structural plasticity. IEEE transactions on nanobioscience **17**(4) (2018) 560–566
41. Păun, A., Păun, G.: Small universal spiking neural p systems. BioSystems **90**(1) (2007) 48–60
42. Cabarle, F.G.C., de la Cruz, R.T.A., Adorna, H.N., Dimaano, M.D., Peña, F.T., Zeng, X.: Small spiking neural p systems with structural plasticity. Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday (2018) 45–56

43. de la Cruz, R.T.A., Cabarle, F.G.C., Adorna, H.N.: Steps toward a homogenization procedure for spiking neural p systems. Theoretical Computer Science **981** (2024) 114250
44. de la Cruz, R.T.A., Cabarle, F.G.C., Macababayao, I.C.H., Adorna, H.N., Zeng, X.: Homogeneous spiking neural p systems with structural plasticity. Journal of Membrane Computing **3** (2021) 10–21
45. Cabarle, F.G.C., Hernandez, N.H.S., Martínez-del Amor, M.Á.: Spiking neural p systems with structural plasticity: Attacking the subset sum problem. In: International Conference on Membrane Computing, Springer (2015) 106–116
46. Paul, P., Sosik, P.: Solving the sat problem using spiking neural p systems with coloured spikes and division rules. (2024)
47. Ishdorj, T.O., Leporati, A.: Uniform solutions to sat and 3-sat by spiking neural p systems with pre-computed resources. Natural Computing **7** (2008) 519–534
48. Idowu, R.K., Chandren, R., Othman, Z.A.: Advocating the use of fuzzy reasoning spiking neural p system in intrusion detection. In: Asian Conference on Membrane Computing ACMC 2014. (2014) 1–5
49. Song, T., Pang, S., Hao, S., Rodríguez-Patón, A., Zheng, P.: A parallel image skeletonizing method using spiking neural p systems with weights. Neural Processing Letters **50** (2019) 1485–1502

Virus Machines And Their Matrix Representations

Antonio Ramírez-de-Arellano[1,2], Francis George C. Cabarle[1,2,3], David Orellana-Martín[1,2], Henry N. Adorna[3], Mario J. Pérez-Jiménez[1,2]

[1]Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[2]SCORE lab, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: aramirezdearellano@us.es, dorellana@us.es, fcabarle@us.es, marper@us.es
[3]Department of Computer Science, University of the Philippines Diliman, 1101 Quezon City, Philippines
E-mail: fccabarle@up.edu.ph, hnadorna@up.edu.ph

Summary. In this work, we present an extension of the matrix representation for virus machines. Structures such as vectors and matrices are useful in practical and theoretical domains. Given the matrix representation of virus machines, the computations of such machines can be expressed in terms of linear algebra operations. Previously, the matrix representation was for deterministic machines only. Presently, we provide a virus machine to nondeterministically generate the set of all natural numbers. We use the virus machine for generating natural numbers to demonstrate the extension of the matrix representation. Finally, we give some conclusions and directions for further work.

Keywords: Virus machine, Matrix representation, Natural computing.

1 Introduction

Virus machines (in short, VMs) are unconventional and bio-inspired models of computing introduced in [1], inspired by the transmission of viruses in a network of hosts. Since their introduction, VMs have been shown to be Turing complete, that is, they can generate, accept, or compute functions over computable sets of numbers [1, 2]. Arithmetic and pairing functions, as well as simulations of workflow patterns have been investigated in the context of VMs [3, 4, 5].

Briefly, a VM is a heterogeneous graph consisting of 3 subgraphs: *host* graph, containing *hosts as nodes*, with *directed and weighted edges* among hosts as *channels*, where hosts contain zero or more copies of the virus object v; *instruction* graph, where *nodes are instructions* to be activated, with directed and

weighted edges between nodes identify which instruction to next activate; *channel-instruction* graph, which connects one instruction node to at most one channel between hosts. By default all channels between hosts are closed: if an instruction i node is activated and i connects to a channel, the channel is opened and viruses are transferred from the source to the destination host.

Previously in [6] a matrix representation for VMs was introduced with the following idea, at some time instant t: the *configuration*, that is, the number of virus objects in each host is represented by a vector; an instruction vector represents the instruction activated at t; a virus transmission matrix of the VM dictates the effect of each instruction (written as rows) to each host (written as columns). Computations of the VM, that is, transitions from one configuration to the next, are provided by linear algebra operations of such a representation.

Other bio-inspired models with matrix representations are spiking neural P systems (SN P systems, in short) [7, 8], including other variants and optimisations in [9, 10, 11, 12]. The matrix representations of SN P systems are used in their automatic design, simulations, and verifications, see for instance [13, 11, 14, 15].

The present work extends the matrix representation of virus machines from [6]. A new VM Π_{Nat} for generating the set of natural numbers is presented. The matrix representation from [6] applies to deterministic VMs only, while the present work extends it to nondeterministic VMs. The VM Π_{Nat} is used to demonstrate the extension with the nondeterministic semantic.

The present work is organised as follows. Section 2 provides the basic definition, syntax, and semantics of VMs. A VM to perform addition, and a new and nondeterministic VM Π_{Nat} are included in Section 2. Section 3 defines the matrix representation first for deterministic VMs, followed by the extensions of the representation for nondeterministic VMs. Conclusions and ideas for further work are provided in Section 4.

2 Virus Machines: Brief definition

In [1], Virus Machines were introduced as a universal model of computation, in the sense that they can calculate every set computable by a Turing machine. While the formal definition can be followed in the founding work, we remark on the syntax here, while the semantics will be explained with an explicit example.

First, let us formally define the **syntax** of virus machines.

Definition 1. *Let a virus machine Π of degree (p, q) with $p, q \geq 0$ defined as:*

$$\Pi = (\Gamma, H, I, D_H, D_H, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

where:

- $\Gamma = \{v\}$ *is the singleton alphabet.*

- $H = \{h_1, \ldots, h_p\}$ *is the ordered set of hosts,* h_{out} *can be either in* H *or not (for this work, we will suppose always* $h_{out} \notin H$*,* $I = \{i_1, \ldots, i_q\}$ *the ordered set of instructions.*
- $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ *is the weighted and directed (WD) host graph, where the edges are called channels and* $w_H : H \times H \cup \{h_{out}\} \to \mathbb{N}$*.*
- $D_I = (I, E_I, w_I)$ *is the WD instruction graph and* $w_I : I \times I \to \{1, 2\}$*.*
- $G_C = (E_H \cup I, E_I)$ *is a unweighted bipartite graph called channel-instruction graph, where the partition associated is* $\{E_H \cup I\}$*.*
- $n_1, \ldots, n_p \in \mathbb{N}$ *are the initial number of viruses in each host* $h_1, \ldots, h_p$*, respectively.*

Regarding the **semantics**, a *configuration* or an *instantaneous description* at an instant $t \geq 0$ is the tuple $\mathcal{C}_t = (a_{1,t}, a_{2,t}, \ldots, a_{p,t}, u_t, a_{0,t})$ where for each $j \in \{1, \ldots, p\}$, $a_{j,t} \in \mathbb{N}$ represents the number of viruses in the host h_j at instant t, and $u_t \in I \cup \{\#\}$. To clarify the notation, in this work it will be said as instantaneous description and $\mathcal{C}_t$ will be noted as ID_t, being $ID_0 = (n_1, \ldots, n_p, i_1, 0)$ the *initial instantaneous description.*

From an instantaneous description ID_t, ID_{t+1} is obtained as follows. The instruction that will be activated is u_t if $u_t \in I$, otherwise ID_t is a *halting configuration.* Let us suppose that $u_t \in I$ and that it is attached to the channel $(h_j, h_{j'}) \in E_H$ with weight $w \in \mathbb{N}$, then the channel is *opened* and two possibilities holds:

- If $a_{j,t} > 0$, then there is *virus transmission*, that is, one virus is consumed from h_j and is sent to the host $h_{j'}$ replicated by w. The next activated instruction will follow the highest weight path in the instruction graph. In case the highest path is not unique, it is chosen nondeterministically. In case there is no possible path, then $u_{t+1} = \#$
- If $a_{j,t} = 0$, then there is no virus transmission and the next instruction follows the least weight path. For the other cases, it is analogous to the previous assumption.

To clarify the behavior of these devices, we will show two specific examples. The first one will be deterministic, and the other one will be nondeterministic. These two examples will be reused for the following section. Before this, some brief explanations of the function computing and number generating modes are presented. For a more formal and detailed definition we refer to [4, 2].

2.1 Virus Machines computing functions: Addition function

A *virus machine with input of degree* (p, q, r) with $p, q \geq 1$ and $1 \leq r \leq p$ is defined as:

$$\Pi = (\Gamma, H, H_r, I, D_H, D_H, G_C, n_1, \ldots, n_p, i_1, h_{out}),$$

where $\Pi = (\Gamma, H, I, D_H, D_H, G_C, n_1, \ldots, n_p, i_1, h_{out})$ is a VM of degree (p, q), and $H_r \subseteq H$ is the ordered set of input hosts. For a given input $(a_1, \ldots, a_r) \in \mathbb{N}$,

the initial configuration of $\Pi + (a_1, \ldots, a_r)$ will be the addition to the input hosts the values $a_1, \ldots, a_r$ respectively.

We say a partial function $f : -\mathbb{N}^r \to \mathbb{N}$ is computed by a VM with input of degree (p, q, r) if for each input $\vec{a} \in \mathbb{N}$ well defined in f, all the computations of $\Pi + \vec{a}$ halt and returns $f(\vec{a})$, otherwise all the computations are non-halting computations.

To clarify this, let Π_{add} be a VM [4] with input of degree $(2, 3, 2)$ visually represented in Figure 1. The hosts are drawn as squares, and instructions are drawn as blue dots; the initial amount of viruses at each host is written inside them. For simplicity, the weights of the arcs with weight 1 are omitted. Finally, the instruction-channel graph is represented by red dotted lines.

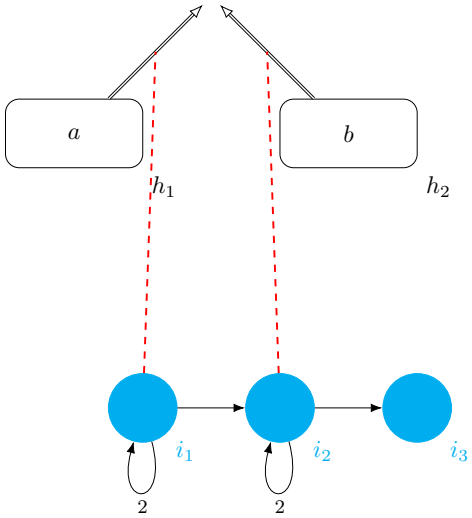


Fig. 1. The VM $\Pi_{Add} + (a, b)$.

A method to formally verify these devices is by looking for **invariants** that highlight relevant loops of the device. For example, two invariants holds in this machine:

$$\varphi(k) \equiv \mathcal{C}_k = (a - k, b, i_1, k), \text{ for each } 0 \leq k \leq a;$$
$$\varphi'(k) \equiv \mathcal{C}_{k+a+1} = (0, b - k, i_2, a + k), \text{ for each } 0 \leq k \leq b;$$

The first invariant φ shows that the a viruses are sent one by one from h_1 to the environment, in this whole process instruction i_1 will be activated, as $\varphi(a)$ is true, the configuration $\mathcal{C}_a = (a - a, b, i_1, a)$ is reached, due to the host h_1 being empty, the next instruction follows the least weight path, that is instruction i_2. Leading the configuration $\mathcal{C}_{a+1} = (0, b, i_2, a)$. At that instant, the second invariant $\varphi'(k)$ is

initialized, which represents the analogous computation, but with host h_2 instead of host h_1. In particular, $\varphi'(b)$ is true, then the following computation holds:

$$\begin{aligned}\varphi'(b) =& \mathcal{C}_{a+b+1} = (0, b-b, i_2, a+b),\\ & \mathcal{C}_{a+b+2} = (0, 0, i_3, a+b), \text{ as } h_2(0),\\ & \mathcal{C}_{a+b+3} = (0, 0, \#, a+b).\end{aligned}$$

Thus, after the $a+b+3$ transition steps, the machine halts and returns $a+b$, which is the addition between (a, b).

2.2 Virus machines generating sets: Natural numbers set

VMs can be defined to generate sets of natural numbers; we say a number $n \in \mathbb{N}$ is generated by a VM Π, if for a computation of Π the machine returns n. We say that a subset $A \subseteq \mathbb{N}$ is generated by a virus machine Π if and only if for every $a \in A$, the number a is generated by Π, and for any halting computation of Π, the output belongs to the set A.

To clarify this, let Π_{nat} be the VM of degree $(2, 4)$ depicted in Figure 2 that generates the set $\mathbb{N} \setminus \{0\}$.

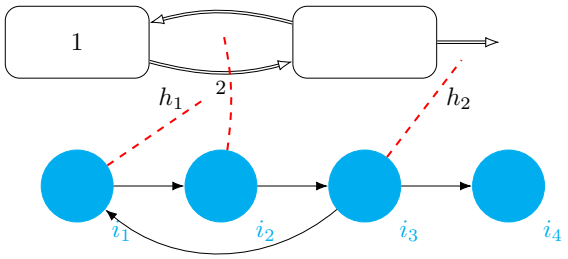


Fig. 2. The VM Π_{Nat}.

To formally prove that the natural numbers set $\mathbb{N}$ is generated, let us see that for each $n \in \mathbb{N} \setminus \{0\}$, there exists a computation of Π_{nat} that generates n. Let us suppose that the number generated is $n \in \mathbb{N} \setminus \{0\}$, then the invariant that holds this machine is:

$$\varphi''(k) \equiv \mathcal{C}_{3k} = (1, 0, i_1, k), \text{ for each } 0 \leq k \leq n-1;$$

The invariant can be easily proved by induction. In particular $\varphi(n)$ is true, thus the following computation holds:

$$\begin{aligned}\varphi''(n-1) \equiv & \mathcal{C}_{3(n-1)} = (1, 0, i_1, n-1),\\ & \mathcal{C}_{3(n-1)+1} = (0, 2, i_2, n-1),\\ & \mathcal{C}_{3(n-1)+2} = (1, 1, i_3, n-1),\\ & \mathcal{C}_{3n} = (1, 0, i_4, n), \text{ nondeterministic decision},\\ & \mathcal{C}_{3n+1} = (1, 0, \#, n),\end{aligned}$$

Thus, after $3n + 1$ transition steps, the machine halts and returns n.

To demonstrate that each halting computation of Π_{nat} is in $\mathbb{N} \setminus \{0\}$, we only have to prove that for any halting computation of Π_{nat}, the output is greater than zero. For this, let us highlight the fact that at the instant $t = 3$, only two possible computations arise: $\mathcal{C}_3 = (1, 0, i_1, 1)$ or $\mathcal{C}_3 = (1, 0, i_4, 1)$. In both cases, one virus has been sent to the environment, as it cannot decrease, the output will be greater than zero. Thus, VM Π_{nat} generates the set $\mathbb{N} \setminus \{0\}$.

3 Matrix Representation

In this section the matrix representation is formally defined for deterministic virus machines, after that, the first ideas on the matrix representation of the nondeterministic virus machines are presented.

3.1 Determinism case

Regarding the semantics of a VM, for any step or instant $t \geq 0$, the *instantaneous description* of Π is $ID_t = (a_{1,t}, a_{2,t}, \ldots, a_{p,t}, u_t, a_{0,t})$, where each $a_{i,t}$ is the number of viruses in the host h_i, the instruction u_t is next activated, and the environment contains $a_{0,t}$ viruses.

For the following definitions, consider a VM Π of degree (p, q), with the notation fixed above and in Definition 1, at any instant t of its computation. We note that definitions and results in the present section for the deterministic case are from [6].

Definition 2 ([6]). *We define the* **configuration vector** *as the vector*

$$\overrightarrow{c}_t = \langle a_{1,t}, a_{2,t}, \ldots, a_{0,t} \rangle,$$

and the **instruction vector** *as the vector*

$$\overrightarrow{i}_t = \langle r_{1,t}, r_{2,t}, \ldots, r_{q,t} \rangle,$$

where $r_{m,t} = 1$ if $u_t = i_m \in I$, otherwise $r_{m,t} = 0$, for $1 \leq m \leq p$. That is, if the activated instruction is $i_j \in I$, then the component $r_{j,t}$ is the only non-zero element of $\overrightarrow{i}_t$.
In particular, vector $\overrightarrow{c}_0 = \langle n_1, n_2, \ldots, n_p\ 0 \rangle$, and $\overrightarrow{i}_0 = \langle 1, 0, \ldots, 0 \rangle$, is the **initial configuration vector** *and* **initial instruction vector**, *respectively.*

Definition 3 ([6]). *A* **virus transmission matrix** *of Π is defined as*

$$M_{\Pi} = [m_{k,j}]_{q \times p+1},$$

where

$$m_{k,j} = \begin{cases} -1, & \text{if instruction } i_k \text{ activates to remove a virus from host } h_j, \\ w, & \text{if } h_j \text{ (or the environment) receives } w \text{ viruses when } i_k \text{ activates}, \\ 0, & \text{otherwise.} \end{cases}$$

Let us apply Definition 2 and Definition 3, to the deterministic Π_{Add} in Figure 1 of Section 2.1. We have $\overrightarrow{c}_0 = \langle a, b, 0 \rangle$ and $\overrightarrow{i}_0 = \langle 1, 0, 0 \rangle$, to mean the following: hosts h_1 and h_2 have a and b viruses, respectively, and the environment is empty, with instruction i_1 first activated. The virus transmission matrix $M_{\Pi_{Add}}$ of Π_{Add} is given by Equation 1.

$$M_{\Pi_{Add}} = \begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{1}$$

The idea of the virus transmission matrix $M_{\Pi_{sub}}$ is to show the effects of the instructions (the rows) to the hosts and environment (the columns). For instance, row 1 of $M_{\Pi_{Add}}$ shows that i_1 has no effect (hence the 0 element) on column 2 (for h_2). The effect of i_1 is to remove 1 and add 1 virus each to h_1 and the environment, respectively. Similarly, row 2 shows that i_2 removes and adds one virus to h_2 and the environment, respectively, but has no effect on h_1. Lastly, i_3 leads to halting as no other instructions follow it.

Definition 4 ([6]). *The* ***instruction control matrices*** *are matrices defined as*

$$M_{I,1} = [a_{k,j}]_{q,q}, \text{ and } M_{I,2} = [b_{k,j}]_{q,q},$$

where

$$a_{k,j} = \begin{cases} 1, & \text{if } (i_k, i_j) \in E_I \text{ and } w_I((i_k, i_j)) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

$$b_{k,j} = \begin{cases} 1, & \text{if } (i_k, i_j) \in E_I \text{ and } w_I((i_k, i_j)) = 2, \\ 0, & \text{otherwise.} \end{cases}$$

To obtain the **configuration transition equation** and the **instruction transition equation**, we need to compute some partial results. Depending on the existence or not of a virus in the origin host, the next configuration is changed or not, respectively.

Define the following *partial configuration* vectors

$$\overrightarrow{c}'_t = \overrightarrow{i}_t \cdot M_{\Pi}, \overrightarrow{c}''_t = \overrightarrow{c}_t + \overrightarrow{c}'_t, \overrightarrow{c}'''_t = \overrightarrow{c}''_t I_{p+1 \times p+2} \tag{2}$$

where $I_{p+1 \times p+2}$ is the identity matrix with $p+1$ rows and $p+2$ columns, since there is one column more than rows, the last column is filled with zeros. The idea

behind each vector is: $\overrightarrow{c}'_t$ is the vector to be subtracted from $\overrightarrow{c}_t$ if there was a virus in the origin host. $\overrightarrow{c}''_t$ is the result of the subtraction between $\overrightarrow{c}_t$ and $\overrightarrow{c}'_t$. If there exists a -1, then it means that there were no viruses present in the origin host. We extend the vector $\overrightarrow{c}''_t$ with one more zero in the vector $\overrightarrow{c}'''_t$ for the following technical detail: Let $m_t = min(\overrightarrow{c}'''_t)$ the **control coefficient at instant t**, then m_t is 0 if there was at least one virus in the origin host and -1 otherwise. To obtain the next configuration vector we have the following result.

Theorem 1 ([6]). *Let Π be a VM with q instructions and p hosts, M_Π is the virus transmission matrix, $\overrightarrow{c}_t$ and $\overrightarrow{i}_t$ are the configuration and instruction vectors at instant t, respectively. We obtain the next configuration vector $\overrightarrow{c}_{t+1}$ using the following* **transition equation***:*

$$\overrightarrow{c}_{t+1} = \overrightarrow{c}_t + (1 + m_t)\overrightarrow{c}'_t.$$

Let us apply Definition 4, Theorem 1 to $M_{\Pi_{Add}}$. Given $\overrightarrow{c}_0 = \langle a, b, 0\rangle$ and $\overrightarrow{i}_0 = \langle 1, 0, 0\rangle$ we have

$$\overrightarrow{c}'_0 = \overrightarrow{i}_0 \cdot M_{\Pi_{sub}} = \langle -1, 0, 1\rangle, \overrightarrow{c}''_0 = \overrightarrow{c}_0 + \overrightarrow{c}'_0 = \langle (a-1), b, 1\rangle,$$

with $\overrightarrow{c}'''_0 = \overrightarrow{c}''_0 \cdot I_{3\times 4} = \langle (a-1), b, 1, 0\rangle$. We also have $m_0 = min((a-1), b, 1, 0, 0)$ so that if $a > 0$ we have $m_0 = 0$. The *next configuration* of Π_{Add} is

$$\overrightarrow{c}_1 = \overrightarrow{c}_0 + (1 + m_0) \cdot \overrightarrow{c}'_0 = \langle a, b, 0\rangle + \overrightarrow{c}'_0 = \langle (a-1), b, 1\rangle.$$

Let us now move to definitions to obtain the equation for the next instruction, using m_t defined above.
Let the *partial instruction* vectors and *control sum* be

$$\overrightarrow{i}_{t,1} = \overrightarrow{i}_t \cdot M_{I,1},\ \overrightarrow{i}_{t,2} = \overrightarrow{i}_t \cdot M_{I,2},\ \overrightarrow{i}'_t = \overrightarrow{i}_{t,1} + 2\overrightarrow{i}_{t,2}, s_t = i'_t \cdot 1_{q\times 1} \quad (3)$$

s_t is the **control sum at instant** t, being $1_{q\times 1}$ a column vector with q ones. s_t is a scalar number that has 4 possible values:

$$s_t = \begin{cases} 0, & \text{if current instruction } \overrightarrow{i}_t \text{ has no next instructions,} \\ 1, & \text{if current instruction } \overrightarrow{i}_t \text{ has one next instruction,} \\ 2, & \text{if current instruction } \overrightarrow{i}_t \text{ has two next instructions,} \\ & \text{both of them with an arc of weight 1,} \\ 3, & \text{if current instruction } \overrightarrow{i}_t \text{ has two next instructions, one with an arc} \\ & \text{of weight 1 and one with an arc of weight 2,} \end{cases} \quad (4)$$

If we *restrict the VM Π to be deterministic* (that is, $s_t \neq 2$), we can define the next instruction i_{t+1} as follows:

$$\begin{aligned}\overrightarrow{i}_{t+1} = \frac{(1-s_t)(2-s_t)(3-s_t)}{6}\overrightarrow{i}_{t,1} + \frac{(-s_t)(2-s_t)(3-s_t)}{-2}\overrightarrow{i}_{t,1} + \\ + \frac{(-s_t)(1-s_t)(2-s_t)}{-6}(m_t \cdot \overrightarrow{i}_{t,1} + (1+m_t)\overrightarrow{i}_{t,2})\end{aligned} \tag{5}$$

If we simplify the terms for $\overrightarrow{i}_{t+1}$, the following result provides the next instruction to be activated.

Theorem 2 ([6]). *Let Π be a VM of degree (p, q), M_Π the virus transmission matrix, $M_{I,1}$ and $M_{I,2}$, the instructions control matrices, $\overrightarrow{c}_t$ and $\overrightarrow{i}_t$ are the configuration and instruction vectors at instant t, respectively. We obtain the next configuration vector $\overrightarrow{i}_{t+1}$ using the following* **instruction control equation***:*

$$\overrightarrow{i}_{t+1} = \frac{2-s_t}{6}(((m_t 2)s_t^2 + (5-m_t)s_t + 3)\overrightarrow{i}_{t,1} + s_t(1-s_t)(1+m_t)\overrightarrow{i}_{t,2}), \tag{6}$$

where $\overrightarrow{i}_{t,j} = M_{I,j}\overrightarrow{i}_t$, for $j \in \{1, 2\}$, m_t and s_t are the control coefficient and the control sum at instant t, respectively.

Remark 1. Theorem 2 is true if and only if Π is deterministic.

Let us apply the partial instruction vectors, s_t, and Theorem 2 to $M_{\Pi_{Add}}$. Now $s_0 = 3$ from equation 4 since the two arcs of i_1 has a sum of 3 for their weights. Due to $s_0 = 3$ only the rightmost term of equation 5 is nonzero, and more specifically the term with $\overrightarrow{i}_{0,2}$, providing $\overrightarrow{i}_1 = 1 \cdot \overrightarrow{i}_{0,2} = \langle 2, 0, 0\rangle$. The *next instruction* to be activated is i_1 again due to $a > 0$ at instant $t = 0$.

3.2 Non-determinism case

As we stated in Remark 1, the definitions and results presented in the previous subsection are for deterministic virus machines, however, this computing paradigm develops nondeterministic computing models, so it should be taken into account. In this subsection we develop the first ideas on this purpose.

First, Theorem 1 remains true for nondeterministic behavior, as the nondeterminism comes from the instruction that will be activated in the following step. Because of this, we will focus on Theorem 2.

For being the nondeterministic case at an instant t, it means that the highest/least weight path is not unique, that is the case $s_t = 2$, where the two possible next instructions have an arc of weight 1. In addition, $\overrightarrow{i}_{t,1}$ has two non-zero components and $\overrightarrow{i}_{t,2}$ is the zero vector. Keeping the same notation as before and applying Theorem 2 the following equation holds:

$$\overrightarrow{i}_{t+1} = \frac{2-s_t}{6}(((m_t 2)s_t^2 + (5-m_t)s_t + 3)\overrightarrow{i}_{t,1} + s_t(1-s_t)(1+m_t)\overrightarrow{i}_{t,2}),$$

Evaluating the variables we have that $\overrightarrow{i}_{t+1}$ is the zero vector. To alleviate this problem, we propose an extension of the equation by adding the following term $\frac{s_t(1-s_t)(3-s_t)}{-2}\overrightarrow{i}_{t,1}$. Thus, the following Theorem holds:

Theorem 3. *Let Π be a VM of degree (p, q), M_Π the virus transmission matrix, $M_{I,1}$ and $M_{I,2}$, the instructions control matrices, $\overrightarrow{c}_t$ and $\overrightarrow{i}_t$ are the configuration and instruction vectors at instant t, respectively. We obtain the next configuration vector $\overrightarrow{i}_{t+1}$ using the following* **auxiliary instruction control equation***:*

$$\begin{aligned}\overrightarrow{i}''_{t+1} = \frac{2-s_t}{6}(((m_t 2)s_t^2 + (5-m_t)s_t + 3)\overrightarrow{i}_{t,1} + s_t(1-s_t)(1+m_t)\overrightarrow{i}_{t,2}) + \\ + \frac{s_t(1-s_t)(3-s_t)}{-2}\overrightarrow{i}_{t,1},\end{aligned} \tag{7}$$

where $\overrightarrow{i}_{t,j} = M_{I,j}\overrightarrow{i}_t$, for $j \in \{1, 2\}$, m_t and s_t are the control coefficient and the control sum at instant t, respectively.

The vector $\overrightarrow{i}''_{t+1}$ is a binary vector that can be written as $\overrightarrow{i}''_{t+1} = \sum_{k\in K} e_k$, where $K \subseteq \{1, \ldots, q\}$, and e_k is the corresponding euclidean basis vector. Then **instruction control equation** *holds:*

$$\overrightarrow{i}_{t+1} = e_{k'},$$

where k' is nondeterministically chosen from the set K.

Let us show the example presented in Subsection 2.2 to clarify this. First, we have $\overrightarrow{c}_0 = \langle 1, 0, 0\rangle$ and $\overrightarrow{i}_0 = \langle 1, 0, 0, 0\rangle$. The virus transmission matrix $\mathcal{M}_{\Pi_{nat}}$ is given by Equation 8.

$$M_{\Pi_{nat}} = \begin{pmatrix} -1 & 2 & 0 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{8}$$

As we said in the Subsection 2.2, at instant 3 a nondeterministic decision is made, let us see how it works with the matrix representation. For that, let us see the instant 2, we have $\overrightarrow{i}_2 = \langle 0, 0, 1, 0\rangle$, and $\overrightarrow{c}_2 = \langle 1, 1, 0\rangle$. From here we have the following:

$$\begin{aligned}\overrightarrow{c}'_2 &= \langle 0, -1, 1\rangle, \\ \overrightarrow{c}''_2 &= \langle 1, 0, 1\rangle, \\ \overrightarrow{c}'''_2 &= \langle 1, 0, 1, 0\rangle.\end{aligned}$$

Thus, $m_2 = 0$. By Theorem 1, we have:

$$\overrightarrow{c}_3 = \overrightarrow{c}_2 + (1 + m_t)\overrightarrow{c}_2 = \langle 1, 0, 1\rangle.$$

What is new here is how we obtain the following instruction vector. Here we have the following:

$$\begin{aligned}\overrightarrow{i}_{2,1} &= \langle 1, 0, 0, 1\rangle,\\ \overrightarrow{i}_{2,2} &= \langle 0, 0, 0, 0\rangle,\\ \overrightarrow{i}'_2 &= \langle 1, 0, 0, 1\rangle,\\ s_2 &= 2.\end{aligned}$$

By the Theorem 3 we have $\overrightarrow{i}''_3 = \overrightarrow{i}_{2,1}$, which can be written as $\overrightarrow{i}''_3 = e_1 + e_4 = \langle 1, 0, 0, 0\rangle + \langle 0, 0, 0, 1\rangle$. Thus, a nondeterministic decision arises choosing $\overrightarrow{i}_3 = e_1$ or $\overrightarrow{i}_3 = e_4$. That represents, exactly, the nondeterministic decision of going to i_1 or i_4 as expected.

4 Conclusion

In recent years, transforming or representing computing processes in linear algebra operations has been a major scope because of their efficient implementations. In this work, a matrix representation of virus machines has been presented with two explicit examples, one with only the deterministic behavior, and the other with nondeterministic behavior. It is interesting to note that this representation opens an interesting framework for the invariants, which is crucial in the formal verification of these devices.

A next direction is to apply the representation in this work to the simulation of workflow patterns in [5]. Such patterns have been studied previously in the framework of spiking neural P systems (in short, SN P systems), see for instance [16, 17, 18]. In order to represent VMs for such patterns the representation needs to be extended to (instruction or channel) parallel VMs, another interesting direction. The matrix representation of VMs, perhaps with a corresponding implementation in software, can help in the verification of the simulated patterns.

The matrix representation in [6] and extended in the present work further opens the simulation in massively parallel processors, such as graphics processing units (in short, GPUs). GPUs are also known as *accelerators* due to their more optimised performance with linear algebra structures, compared to CPUs. For instance, many P system simulations benefit from the use of GPUs [19]. More specifically, the matrix representations of SN P systems, see for instance [7], result in further optimisations in their GPU simulations [10, 14]. For instance the ideas from [11] are experimentally validated in [20], with larger and exhaustive tests [21] which outperform state-of-the-art GPU software libraries.

It is also interesting to continue investigating *reachability*, other static or dynamic properties of VMs, and the complexity of deciding such properties. For

instance, given some configuration $\overrightarrow{c}$ is a configuration $\overrightarrow{c}'$ reachable, where $\overrightarrow{c} \neq \overrightarrow{c}'$? That is, is there a sequence of transitions starting from $\overrightarrow{c}$ and ends with $\overrightarrow{c}'$? The matrix representation can help with this problem, as well as other ideas such as liveness, deadness, coverability [22, 23].

Acknowledgements

F.G.C. Cabarle is supported by the QUAL21 008 USE project, "Plan Andaluz de Investigación, Desarrollo e Innovación" (PAIDI) 2020 and "Fondo Europeo de Desarrollo Regional" (FEDER) of the European Union, 2014-2020 funds. A. Ramírez-de-Arellano is supported by the Zhejiang Lab BioBit Program (Grant No. 2022BCF05).

References

1. Chen, X., Pérez-Jiménez, M.J., Valencia-Cabrera, L., Wang, B., Zeng, X.: Computing with viruses. Theoretical Computer Science **623** (2016) 146–159
2. Ramírez-de-Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Generating, computing and recognizing with virus machines. Theoretical Computer Science **972** (07 2023) 114077
3. Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Basic arithmetic calculations through virus-based machines. In Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Adeli, H., eds.: Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence, Cham, Springer International Publishing (2022) 403–412
4. Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Using virus machines to compute pairing functions. International Journal of Neural Systems **33**(05) (2023) 2350023
5. Ramírez-de-Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Virus machines at work: Computations of workflow patterns. International Summer Conference (ISC 24) of Decision Science Alliance, 6-7 June 2024 València, Spain
6. Ramírez-de Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Pérez-Jiménez, M.J., Adorna, H.N.: Matrix representation of virus machines. In Ferrández Vicente, J.M., Val Calvo, M., Adeli, H., eds.: Bioinspired Systems for Translational Applications: From Robotics to Social Engineering, Cham, Springer Nature Switzerland (2024) 420–429
7. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural P systems. In: Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers 11, Springer (2011) 377–391
8. Adorna, H.N.: Matrix representations of spiking neural P systems: Revisited. arXiv preprint arXiv:2211.15156 (2022)

9. Jimenez, Z.B., Cabarle, F.G.C., de la Cruz, R.T.A., Buño, K.C., Adorna, H.N., Hernandez, N.H.S., Zeng, X.: Matrix representation and simulation algorithm of spiking neural p systems with structural plasticity. Journal of Membrane Computing **1** (2019) 145–160
10. Carandang, J.P., Cabarle, F.G.C., Adorna, H.N., Hernandez, N.H.S., Martínez-del Amor, M.Á.: Handling non-determinism in spiking neural P systems: Algorithms and simulations. Fundamenta Informaticae **164**(2-3) (2019) 139–155
11. Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural p systems with sparse matrix-vector operations. Processes **9**(4) (2021) 690
12. Ballesteros, K.J., Cailipan, D.P.P., de la Cruz, R.T.A., Cabarle, F.G.C., Adorna, H.N.: Matrix representation and simulation algorithm of numerical spiking neural p systems. Journal of Membrane Computing **4**(1) (2022) 41–55
13. Aboy, B.C.D., Bariring, E.J.A., Carandang, J.P., Cabarle, F.G.C., De La Cruz, R.T., Adorna, H.N., Martínez-del Amor, M.Á.: Optimizations in cusnp simulator for spiking neural p systems on cuda gpus. In: 2019 International Conference on High Performance Computing & Simulation (HPCS), IEEE (2019) 535–542
14. Gungon, R.V., Hernandez, K.K.M., Cabarle, F.G.C., De la Cruz, R.T.A., Adorna, H.N., Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I.: Gpu implementation of evolving spiking neural p systems. Neurocomputing **503** (2022) 140–161
15. 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) (2021) 133–148
16. Cabarle, F.G.C., Adorna, H.N.: On structures and behaviors of spiking neural p systems and petri nets. In: Membrane Computing: 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers 13, Springer (2013) 145–160
17. Cabarle, F.G.C., Buño, K.C., Adorna, H.N.: Time after time: Notes on delays in spiking neural p systems. In Nishizaki, S.y., Numao, M., Caro, J., Suarez, M.T., eds.: Theory and Practice of Computation, Tokyo, Springer Japan (2013) 82–92
18. Song, T., Zeng, X., Zheng, P., Jiang, M., Rodriguez-Paton, A.: A parallel workflow pattern modeling using spiking neural p systems with colored spikes. IEEE transactions on nanobioscience **17**(4) (2018) 474–484
19. Martínez-del-Amor, M.A., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Simulating P systems on GPU devices: a survey. Fundamenta Informaticae **136**(3) (2015) 269–284
20. Hernández-Tello, J., Martínez-Del-Amor, M.Á., Orellana-Martín, D., Cabarle, F.G.: Sparse matrix representation of spiking neural p systems on gpus. In Vaszil, G., Zandron, C., Zhang, G., eds.: Proc. International Conference on Membrane Computing (ICMC 2021), Chengdu, China and Debrecen, Hungary, 25 to 26 August 2021 (Online). (2021) 316–322
21. Hernández-Tello, J., Martínez-Del-Amor, M.Á., Orellana-Martín, D., Cabarle, F.G.C.: Sparse spiking neural-like membrane systems on graphics processing units. International Journal of Neural Systems **34**(7) (2024) 2450038–2450038
22. Peterson, J.L.: Petri nets. ACM Computing Surveys (CSUR) **9**(3) (1977) 223–252
23. Cabarle, F.G.C., Adorna, H.N.: On structures and behaviors of spiking neural p systems and petri nets. In: Membrane Computing: 13th International Conference,

CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers 13, Springer (2013) 145–160

Virus Machines: To tree or not to tree

Antonio Ramírez-de-Arellano[1,2], Francis George C. Cabarle[1,2,3], David Orellana-Martín[1,2], Henry N. Adorna[3], Mario J. Pérez-Jiménez[1,2]

[1]Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[2]SCORE lab, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: aramirezdearellano@us.es, dorellana@us.es, fcabarle@us.es, marper@us.es
[3]Department of Computer Science, University of the Philippines Diliman, 1101 Quezon City, Philippines
E-mail: fccabarle@up.edu.ph, hnadorna@up.edu.ph

Summary. In the present work we further study the computing power of virus machines, or VMs in short. VMs are computing models inspired by the transmission networks of viruses. VMs consist of hosts that contain zero or more virus objects, and an instruction graph that controls the transmissions of virus objects among hosts. The present work improves the understanding of the computing power of VMs by introducing normal forms. Normal forms restrict the features or the number of such features in a given computing model. For VMs we restrict in our normal forms the features such as the number of hosts, number of instructions, and the number of virus objects in each host. After we recall some known results on the computing power of VMs we give our normal forms. For instance we show characterisations from previous inclusions regarding the computation of finite sets of numbers. We also show new characterisations and normal forms for singleton sets and finite sets. Another result using a new normal form are characterisations when the instruction graphs of VMs are (not) restricted to tree graphs. New characterisations of finite sets from VMs with tree instruction graphs are provided, with some conjectures or open problems.

Keywords: Virus machines, Computational power, Natural computing, normal forms.

1 Introduction

In the present work, we consider some normal forms for virus machines, in short, VMs. Virus machines introduced in [1] are unconventional and natural computing models inspired by networks of virus transmissions. More information on unconventional and natural computing is found in [2] and [3], respectively. From [1, 4]

it is shown that VMs are Turing complete, that is, they are algorithms capable of general purpose computations. From such works it is also shown some VMs for computing specific classes of (in)finite sets of numbers.

Virus machines consists of three *subgraphs*: a directed and weighted *host graph* with nodes and edges referred to as *hosts* and *channels*, respectively; a directed and weighted *instruction graph* where nodes are *instructions* and edge weights determine which instruction to prioritise and next activate; an *instruction-channel* graph which connects an edge between instructions and channels in the previous graphs. Hosts contain zero or more virus objects, and activating an instruction means opening a channel since channels are closed by default. Opening a channel means virus objects from one host are transferred to another host.

Briefly, the idea of a normal form for some computing model is to consider restrictions in the model while maintaining its computer power. That is, the consideration of lower bounds for ingredients of a computing model is a natural direction for investigation. For instance a well-known normal form in language theory is the Chomsky normal form, CNF in short, from [5]. Instead of having an infinite number of forms to write rules in a grammar for context-free sets, CNF shows that two forms are enough. Normal forms in unconventional and bio-inspired models include [6], with recent and optimal results in [7], a bibliography in [8], and a recent survey in [9].

The present work contributes the following to the study of virus machines and their computing power. Some normal forms for VMs are provided, such as: providing characterisations (previously were inclusions) for generating families of finite sets; showing new characterisations for finite sets of numbers using restrictions on the number of required hosts, instructions, or viruses; new characterisations are also given for singleton sets of numbers. We also consider a new restriction: limiting or not limiting the instruction graph to be a tree graph, that is, an acylcic graph. We show for instance that some VMs with a tree instruction graph and with some lower bounds on the number of hosts, instructions, and viruses can only compute finite sets. Our results on normal forms are then used to ask new questions regarding other normal forms and restrictions on VMs.

The organisation of the present work is as follows. In Section 2 we recall in a brief the features of VMs used previously in investigating their computing power. Section 2.1 recalls some known results, while Section 2.2 provides new results concerning normal forms of VMs, specifically for computing finite and singleton sets. Section 3 provides new normal forms, for instance, when the instruction graph is restricted to a tree. Lastly, Section 4 provides conclusions, conjectures, and directions for further work.

2 VM with old ingredients

In this section, the computational power of virus machines in generating mode is discussed, from previous works related to some novel results. In this work the syntax and semantics, in addition to a simple explicit example, have been included in

the other work of this volume related to virus machines [10] For further knowledge about the power of virus machines in generating sets we refer to [11, 4, 1]. First, let us fix some notation.

Let $NVM(p, q, n)$ be the family of sets of natural numbers generated by virus machines with at most p hosts, q instructions, and n viruses in each host at any instant of the computation. For unbounded restrictions, they are replaced by a $*$.

2.1 Old results

This subsection is devoted to reviewing results prior to this work regarding the computing power of VMs with respect to certain classes or families of computable numbers.

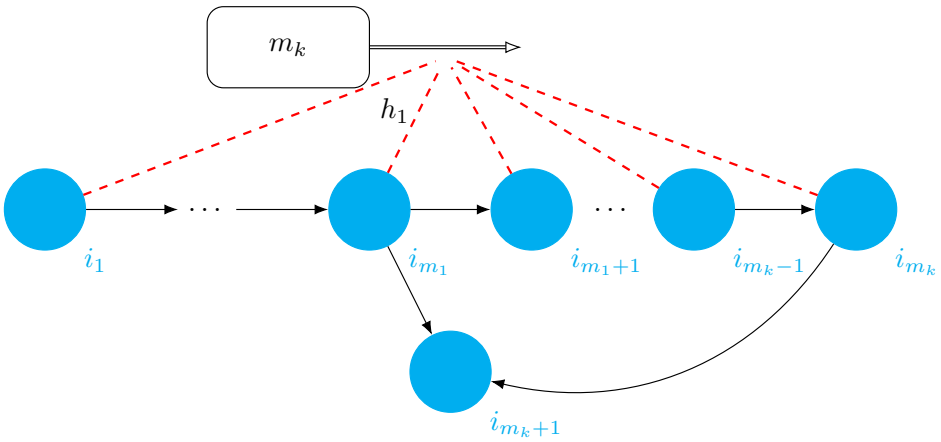


Fig. 1. A virus machine generating $NFIN$ for $NVM(1, *, *)$.

The state-of-the-art is presented in Table 1. The virus machines in the generating mode are Turing Universal; that is, they can generate recursively enumerable sets of numbers (NRE) [1] for unbounded restrictions. This power is severely reduced when the last ingredient is reduced; more precisely, a characterization of semilinear sets ($SLIN$) is proved for $NVM(*, *, 2)$ [11]. From now on, not characterizations but contentions have been proven, for finite sets ($NFIN$) they are contained in $NVM(1, *, *)$ and $NVM(*, *, 1)$ [4]. Finally, the set of power of two numbers is contained in $NVM(2, 7, *)$ [4].

An interesting and natural question is can we further restrict or provide better lower bounds, for known results about VMs? That is, provide "better" characterisations of finite sets or even other families of sets such as the singleton sets, see for instance Table 1. As we focus on finite sets later, let us see the VMs used in [4] to generate finite sets. For $NVM(1, *, *)$ the VM presented in Figure 1, and for

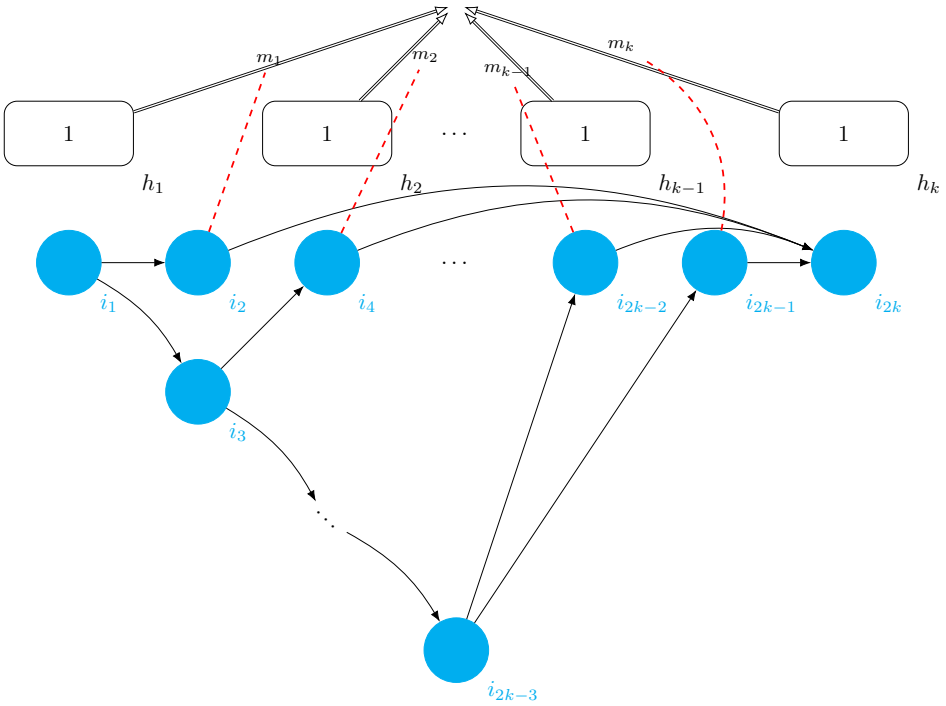


Fig. 2. A virus machine generating $NFIN$ for $NVM(*,*,1)$.

Family of sets	Relation	Hosts	Instructions	Viruses
NRE [1]	$=$	*	*	*
$SLIN$ [11]	$=$	*	*	2
$NFIN$ [4]	$\subset$	1	*	*
	$\subset$	*	*	1
$\{2^n \mid n \geq 0\}$ [4]	$\subset$	2	7	*

Table 1. Previous results: Minimum resources needed for generating family subsets of natural numbers.

$NVM(*,*,1)$ the Figure 2. The corresponding lemmas were called (viruses) and (hosts) respectively, and we follow the same notation in this work.

2.2 New results

Finite sets

Having shown the generation of finite sets by a family of virus machines in the previous section, the bounded ingredient was only one, let us see a family of virus machines with more than one bounded ingredient.

Lemma 1 (Viruses-host). *Let $F = \{m_1, \ldots, m_k\}$ a finite set of natural numbers greater than zero. Then F can be generated by a virus machine of 2 hosts, $2k + 1$ instructions, and the 2 virus in each host at most.*

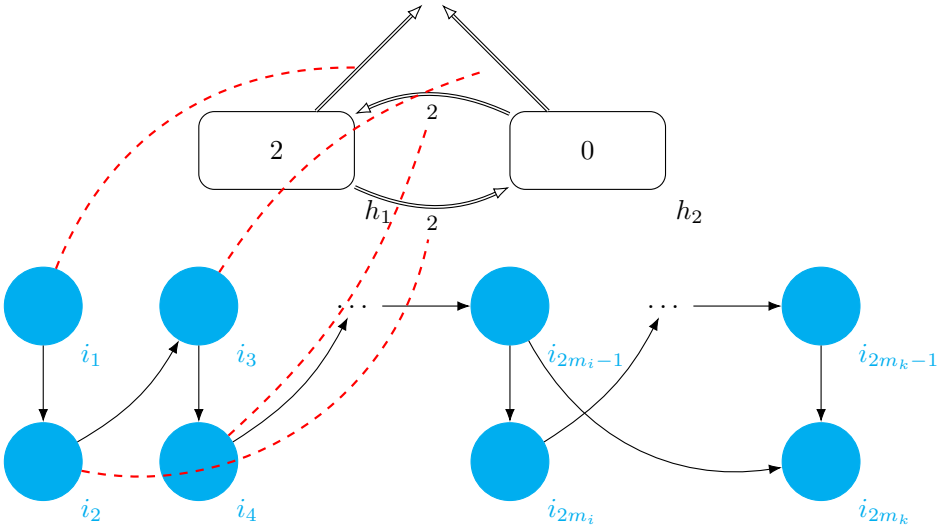


Fig. 3. Virus machine generating the finite set $F = \{m_1, \ldots, m_k\}$.

Proof. Let $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, n_2, \ldots, n_k, i_1, h_{out})$, where:

1. $\Gamma = \{v\}$;
2. $H = \{h_1, h_2\}$;
3. $I = \{i_1, \ldots, i_{2_{m_k}}\}$;
4. $D_H = (H \cup \{h_{out}\}, \{(h_1, h_2), (h_1, h_{out}), (h_2, h_1), (h_2, h_{out})\}, w_H)$, where $w_H((h_1, h_2)) = w_H((h_2, h_1)) = 2$ and $w_H((h_1, h_{out})) = w_H((h_2, h_{out})) = 1$;
5. $D_I = (I, E_I, w_I)$, where $E_I = \{(i_a, i_{a+1}) \mid a \in \{1, \ldots, 2m_k - 1\}\} \cup \{(i_{2m_i-1}, i_{2m_k}) \mid m_i \in F\}$, $w_I((i_j, i_{j'})) = 1 \; \forall (i_j, i_{j'}) \in E_I$;
6. $G_C = (I \cup E_H, E_C)$, where $E_C = \{\{i_{2j+1}, (h_1, h_{out})\}, \{i_{2j}, (h_1, h_2)\} \mid j \in \{0, \ldots, m_k\}, j \text{ even }\} \cup \{\{i_{2j+1}, (h_2, h_{out})\}, \{i_{2j}, (h_2, h_1)\} \mid j \in \{0, \ldots, m_k\}, j \text{ odd }\}$;
7. $n_1 = 2$ and $n_2 = 0$;
8. $h_{out} = h_0$

A visual representation of this virus machine can be found in Figure 3. Let us prove that for each $m_i \in F$, there exists a computation of Π such that it produces m_i viruses in the environment in the halting configuration. Let m_i be the generated number; the following invariant holds:

$$\varphi(x) \equiv \begin{cases} C_{2x} = (2, 0, i_{2x+1}, x) & x \text{ even}, \\ c_{2x} = (0, 2, i_{2x+1}, x) & x \text{ odd}, \end{cases}$$

for each $0 \leq x \leq m_i - 1$. In particular, $\varphi(m_i - 1)$ is true, let us suppose that m_i is odd, then the following computation is verified:

$$\begin{aligned} \mathcal{C}_{2(m_i-1)} &= (2, 0, i_{2(m_i-1)}, m_i - 1), \\ \mathcal{C}_{2m_i} &= (1, 0, i_{2m_k}, m_i), \\ \mathcal{C}_{2m_i+1} &= (1, 0, \#, m_i), \end{aligned}$$

For m_i even the computation is analogous, hence the computation halts in $2m_i + 1$ steps and the number generated is m_i.

Another interesting result is that this inclusion is strict.

Proposition 1. $NFIN \subsetneq NVM(2, *, 2)$.

Proof. Inclusion is direct by the Lemma 1. Let us focus now on the inequality; for that, we construct a virus machine from [10] that generates the set of all natural numbers except the zero, which verifies the restrictions of the proposition.

Let $\Pi_{Nat} = (\Gamma, H, I, D_H, D_I, G_C, 1, 0, i_1, h_{out})$, where:

1. $\Gamma = \{v\}$;
2. $H = \{h_1, h_2\}$;
3. $I = \{i_1, \ldots, i_4\}$;
4. $D_H = (H \cup \{h_{out}\}, \{(h_1, h_2), (h_2, h_{out}), (h_2, h_1)\}, w_H)$, where $w_H((h_1, h_2)) = 2$ and $w_H((h_2, h_{out})) = w_H((h_2, h_1)) = 1$;
5. $D_I = (I, E_I, w_I)$, where $E_I = \{(i_1, i_2), (i_2, _3), (i_3, i_1), (i_3, i_4)\}$, $w_I((i_j, i_{j'})) = 1\ \forall (i_j, i_{j'}) \in E_I$;
6. $G_C = (I \cup E_H, E_C)$, where $E_C = \{\{i_1, (h_1, h_2)\}, \{i_2, (h_2, h_1)\}, \{i_3, (h_2, h_{out})\}\}$;
7. $h_{out} = h_0$;

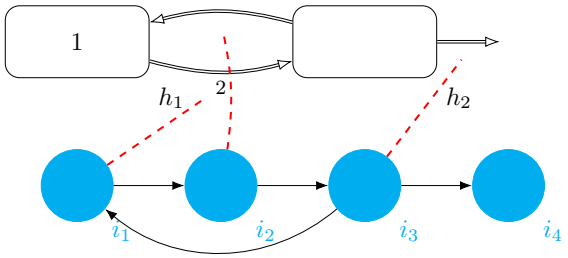


Fig. 4. Virus machine generating the set of natural numbers $\mathbb{N} \setminus \{0\}$.

A visual representation of this virus machine can be found in Fig. 4. Now, let us prove that for each $n \in \mathbb{N}$, there exists a halting computation generating the number n. For generating this number, the following invariant holds:

$$\varphi(k) \equiv \mathcal{C}_{3k} = (1, 0, i_1, k), \text{ for each } 0 \leq k \leq n-1$$

In particular, $\varphi(n-1)$ is true, then the following configuration is verified $\mathcal{C}_{3(n-1)} = (1, 0, i_1, n-1)$, from here, after the 4 transition steps the halting configuration is reached $\mathcal{C}_{3n+1} = (1, 0, \#, n)$, whose output is the natural number n.

With this proposition a new question arises: can we get not only the inclusion but the characterization of the finite sets by a family of virus machines? This is answered in the next section.

Singleton sets

Now let us move to the second family of sets, the Singleton sets, these are sets of natural numbers with only one element, in this work we include the empty set in this family.

Theorem 1. *The following sets of numbers are equivalent to singleton sets:*

1. $NVM(1, *, 1)$;
2. $NVM(*, 1, *)$.

Proof. The proof of equivalence is done by the double inclusion technique.

1. Let us start with the left side inclusion, let $\Gamma = \{v\}$ be a singleton set of natural number $v \in \mathbb{N}$, then it can be generated by the VM Π_{sing_1} of degree $(1, 1)$ depicted in Figure 5, the initial configuration is $\mathcal{C}_0 = (1, i_1, 0)$ and in the following configuration, one virus is consumed and replicated by the weight of the arc, that is v, and sent to the environment, leading to the halting configuration $\mathcal{C}_1 = (0, \#, v)$. Thus, after one transition step, the set generated is $\{v\}$.
   For the reverse inclusion, suppose any VM with only one host and one virus: the host can only be attached to the environment, and let us fix that the weight of that channel is $w \in \mathbb{N}$. Thus, the only number generated is w or none, depending on the instruction graph (if the computation halts or not). Thus, we generate a singleton set.
2. For the inclusion on the left side we can use the VM Π_{sing_1} depicted in Figure 5 as it only has one instruction and the inclusion has already been proven.
   Let us focus on the inclusion of the right side. With only one instruction, there are two possibilities in the instruction graph:
   - The node with a self-arc, which creates an infinite loop, thus a non-halting computation and generating the empty set.

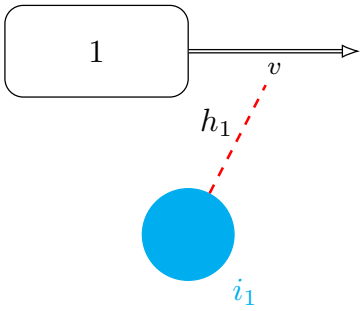


Fig. 5. The VM Π_{sing_1} generating the singleton set $\{v\}$.

- The node with no arcs, thus the machine, halts after only one transition step as there is no other possible path. In this sense, two options can be separated:
  - The instruction is attached to a channel which is attached to the environment,
  - The instruction is not attached to a channel which is attached to the environment, thus the number generated is 0.

3 To tree or not to tree

Until now, the computational power of virus machines was studied by bounding/unbounding the three main ingredients: hosts, instructions, and the number of viruses at any time of computation. Nevertheless, virus machines are heterogeneous networks divided by three graphs, thus more restrictions can be studied, for example, the kinds of graphs that form the instruction graph.

In this section, a novel and interesting scope is proposed to discuss the computational power of virus machines: the *instruction graph properties*.

For this study, several notation and clarifications must first be presented.

Definition 1. *A path in a directed graph $G = (V, E)$ is a sequence of edges $(e_1, \ldots, e_{n-1})$, for which there is a sequence of vertices $(v_1, \ldots, v_n)$, such that $e_i = (v_i, v_{i+1})$, for each $i = 1, \ldots, n-1$, and $v_i \neq v_j$, for all $i, j = 1, \ldots, n$. Under the same conditions, if $(v_n, v_1) \in E$, then the path is called a cycle. A graph without cycles is called a tree. The depth of a tree is the longest path of the tree.*

We say that v_1 is connected to v_n if there is a path $w = (e_1, \ldots, e_{n-1})$, whose sequence of vertices is $(v_1, \ldots, v_n)$. We denote by $V(v_i) \subseteq V$ the subset of vertices that are connected by a path from v_i.

A graph $G = (V, E)$ is connected if there are paths that contain each pair of vertices. A connected component of the graph G is a subgraph graph $G' = (V', E')$, such that $V' \subseteq V$, $E' \subseteq E$ where $(v_i, v_j) \in E'$ if and only if $(v_i, v_j) \in E$, and $v_i, v_j \in V'$ are connected.

Proposition 2 (Invariance). *If the instruction graph D_I of a virus machine Π of degree (p, q), with $p, q \geq 1$,*

$$\Pi = (H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out}),$$

is not connected, then there exists another virus machine Π' of degree (p, q'), with $q' \leq q$, which has the same computation.

Proof. Let Π be the virus machine fixed in the statement, setting the instruction graph to $D_I = (I, E_I, w_I)$, as it is not connected; then $I(i_1) \neq I$. Let Π' be the virus machine of degree $(p, q') = |I(i_1)|)$, defined as Π but with a new instruction graph $D_{I(i_1)} = (I(i_1), E_{I(i_1)}, w_{I(i_1)})$.

Due to the semantics associated with virus machines, any instruction that can be activated must be connected by a path from the initial instruction; thus, the set of instructions of Π that can be activated at some instant of the computation is contained in $I(i_1)$, therefore Π' has the same computation.

Using this result, from now on, all the virus machines defined are supposed to have a connected instruction graph, with the connected component $I(i_1)$, being i_1 the initial instruction. In addition, the same notation of the components of a virus machine Π is used for the following results.

3.1 Instruction graph as a tree

Proposition 3. *If the instruction graph is a tree, then all computations halt. In addition, the number of transition steps is bounded by the depth of the tree.*

Having this restriction on the instruction graph is a limitation of the power of these devices. More precisely, we lose Turing universality; let us see a characterization of finite sets of natural numbers with this restriction. First, let us fix some notation:

Let $NVM_{tree}(p, q, n)$ be the family of sets of natural numbers that can be generated by a virus machine with a tree instruction graph, at most p hosts, q instructions and n virus in each host at any instant of computation. In case there is no restriction, it is written as $*$. Let $NFIN$ be the family of finite sets of natural numbers.

The following results hold:

Corollary 1. $NVM_{tree}(*, *, *) \subseteq NFIN$

Proof. The inclusion $NVM_{tree}(*, *, *) \subseteq NFIN$ is direct by the Proposition 3, as all computations halt, then every virus machine halts in a finite number of steps, thus the set of numbers that can be generated is finite.

Theorem 2. $NVM_{tree}(p, *, *) = NFIN$, *for each* $p \geq 1$.

Proof. The left inclusion $NVM_{tree}(*,*,*) \subseteq NFIN$ is direct from Corollary 1.

The right inclusion $NFIN \subseteq NVM_{tree}(p,*,*)$, for each $p \geq 1$, is proved in Lemma 1 (host) of the work [4], where the virus machine presented was Figure 1, which has the instruction graph as a tree.

Corollary 2. $NVM_{tree}(*,*,*) = NFIN$.

Proof. Direct from Corollary 1 and Theorem 2.

Theorem 3. $NVM_{tree}(*,*,n) = NFIN$, *for each* $n \geq 1$.

Proof. Again, the left inclusion $NVM_{tree}(*,*,*) \subseteq NFIN$ is directly related to the Corollary 1.

The right inclusion $NFIN \subseteq NVM_{tree}(*,*,n)$, for each $n \geq 1$ is proved in Lemma 2 (viruses) of the work [4], where a virus machine with an instruction graph as a tree was presented, generating finite number sets.

Another interesting result occurs if we also bound the amount of hosts:

Theorem 4. $NVM_{tree}(p,*,n) = NFIN$, *for each* $p \geq 2$, *and* $n \geq 2$.

Proof. The right-hand inclusion is by applying again the Corollary 1. For the left side, we can use the Lemma 1 where the VM presented in Figure 3 has the instruction graph as a tree, therefore, $NVM_{tree}(p,*,n) \subseteq NFIN$.

Family of sets	Symbol	Hosts	Instructions	Viruses
$NFIN$ (Corollary 2)	=	*	*	*
(Theorem 2)	=	1	*	*
(Theorem 3)	=	*	*	1
(Theorem 4)	=	2	*	2

Table 2. Minimum resources needed for generating/characterizing family subsets of natural numbers with the *instruction graph as a tree*.

We summarise our main results so far in Table 3.

4 Conjectures and conclusions

In the present work we considered normal forms for VMs: first, by summarising known results for some families of number sets (see Table 1); next, by providing some new results and showing strict characterisations from previous inclusions (see Table 2). We summarise our results and ask new questions for our sequel works regarding normal forms of VMs in Table 3. The rows marked with "?" in Table 3 are open questions, such as if $NVM_{tree}(*,2,*)$ is a strict superset of $NFIN$. It

is also interesting to consider "new" normal forms, such as the graph properties of the host graph and the instruction-channel graph. For instance, considering the weights of the host graph, or if a bijection exists between instructions and channels.

One reason for the interest in normal forms is the consideration of "jumps" in computing power from one family of computable sets to another. For instance in Table 1 we know that VMs with unbounded number of hosts, instructions, and viruses are Turing machines, that is, they are general purpose computers. In Table 3 we see that if we are allowed only one host and one virus in a VM, with an arbitrary number of instructions (the graph is not a tree) then the computing power has a significant jump down to singleton sets. Besides realising such jumps, it is also interesting to realise frontiers or thresholds of the ingredients between families of sets.

Another interesting direction is to consider "small" VMs in the sense of [12, 13]. That is, give lower bounds to the number of instructions or hosts required to maintain a certain computing power, in fact, authors in [14] constructed a small universal VM using 9 hosts and 33 instructions. It would be interesting to study also the lower bound number to compute $NFIN, SLIN$. From Table 1 for instance it is interesting to give better lower bounds for characterisations of NRE and $SLIN$. At least for VMs with instruction graph as a tree, Table 2 provides better lower bounds.

In [15] and its sequel [10] a matrix representation is given for VMs. It is interesting to consider properties in the present work, such as lowering the values for ingredients or restriction to a tree graph, using such a representation. Another direction is to consider the results and conjectures in the present work for VMs in the accepting and function computing modes as in [4], or with parallel VMs as in [16, 17].

Family of sets	Symbol	Hosts	Instructions	Viruses	Tree Inst. Graph
Singleton (Theorem 1)	$=$	1	*	1	No
(Theorem 1)	$=$	*	*	1	No
(Theorem 1)	$=$	*	1	*	No
$NFIN$ (Theorem 2)	$=$	1	*	*	Yes
(Theorem 3)	$=$	*	*	1	Yes
(Theorem 4)	$=$	2	*	2	Yes
(Proposition 1)	$\subsetneq$	2	*	2	**No**
	$\subsetneq$?	*	2	*	Yes
	$\subseteq$?	*	3	*	Yes

Table 3. Summary of the minimum resources needed for generating/characterizing family subsets of natural numbers.

Acknowledgements

F.G.C. Cabarle is supported by the QUAL21 008 USE project, "Plan Andaluz de Investigación, Desarrollo e Innovación" (PAIDI) 2020 and "Fondo Europeo de Desarrollo Regional" (FEDER) of the European Union, 2014-2020 funds. A. Ramírez-de-Arellano is supported by the Zhejiang Lab BioBit Program (Grant No. 2022BCF05).

References

1. Valencia-Cabrera, L., Pérez-Jiménez, M.J., Chen, X., Wang, B., Zeng, X.: Basic virus machines. In: 16th International Conference on Membrane Computing (CMC16). (2015) 323–342
2. Adamatzky, A.: Handbook Of Unconventional Computing (In 2 Volumes). World Scientific (2021)
3. Bäck, T., Kok, J.N., Rozenberg, G.: Handbook of natural computing. Springer, Heidelberg (2012)
4. Ramírez-de-Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Generating, computing and recognizing with virus machines. Theoretical Computer Science **972** (07 2023) 114077
5. Chomsky, N.: On certain formal properties of grammars. Information and control **2**(2) (1959) 137–167
6. Ibarra, O.H., Păun, A., Păun, G., Rodríguez-Patón, A., Sosík, P., Woodworth, S.: Normal forms for spiking neural p systems. Theoretical Computer Science **372**(2-3) (2007) 196–217
7. Macababayao, I.C.H., Cabarle, F.G.C., de la Cruz, R.T.A., Zeng, X.: Normal forms for spiking neural p systems and some of its variants. Information Sciences **595** (2022) 344–363
8. Cabarle, F.G.C., Dela Cruz, R.T.A.: A bibliography of normal forms in spiking neural P systems and variants. In: Bulletin of the International Membrane Computing Society. Volume 12. (12 2021) 89–91
9. Cabarle, F.G.C.: Thinking about spiking neural p systems: some theories, tools, and research topics. Journal of Membrane Computing (2024) 1–20
10. Ramírez-de-Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Pérez-Jiménez, M.J., Adorna, H.N.: Virus machines and their matrix representations. 20th Brainstorming Week on Membrane Computing and First Workshop on Virus Machines, 24–26 January 2024, Sevilla, Spain (2024)
11. Chen, X., Pérez-Jiménez, M.J., Valencia-Cabrera, L., Wang, B., Zeng, X.: Computing with viruses. Theoretical Computer Science **623** (2016) 146–159
12. Cabarle, F.G.C., de la Cruz, R.T.A., Adorna, H.N., Dimaano, M.D., Peña, F.T., Zeng, X.: Small spiking neural p systems with structural plasticity. Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday (2018) 45–56
13. Păun, A., Păun, G.: Small universal spiking neural p systems. BioSystems **90**(1) (2007) 48–60
14. Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Using virus machines to compute pairing functions. International Journal of Neural Systems **33**(05) (2023) 2350023

15. Ramírez-de Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Pérez-Jiménez, M.J., Adorna, H.N.: Matrix representation of virus machines. In Ferrández Vicente, J.M., Val Calvo, M., Adeli, H., eds.: Bioinspired Systems for Translational Applications: From Robotics to Social Engineering, Cham, Springer Nature Switzerland (2024) 420–429
16. Ramírez-de-Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Parallel virus machines. Submitted to Journal of Membrane Computing
17. Ramírez-de-Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Virus machines at work: Computations of workflow patterns. International Summer Conference (ISC 24) of Decision Science Alliance, 6-7 June 2024 València, Spain

Author Index