15th Brainstorming Week on Membrane Computing

Sevilla, January 31 – February 3, 2017

Carmen Graciani
Gheorghe Păun
Agustín Riscos-Núñez
Luis Valencia-Cabrera
Editors

15th Brainstorming Week on Membrane Computing

Sevilla, January 31 – February 3, 2017

Carmen Graciani
Gheorghe Păun
Agustín Riscos-Núñez
Luis Valencia-Cabrera
Editors

RGNC REPORT 1/2017

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2017

ISBN: 978-84-946316-1-0

Edita: Fénix Editora
info@fenixeditora.com
www.fenixeditora.com
Telf. (+34) 954 40 74 36

Impreso en España - Printed in Spain

Preface

The 15th Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from January 31 to February 3, 2017, in the organization of the Research Group on Natural Computing (RGNC) from the Department of Computer Science and Artificial Intelligence of Sevilla University. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and all the next editions took place in Sevilla at the beginning of February, each year.

The program was available at `www.gcn.us.es/15bwmc_program`

In the style of previous meetings in this series, the 15th edition of BWMC 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, conjectures waiting for proofs, followed by an intense cooperation among the 23 participants – see the list in the end of this preface. The efficiency of this type of meetings was again proved to be very high and the present volume illustrates this assertion.

The present volume is meant to be a working instrument, part of the interaction started during the stay of authors in Sevilla, making possible a further cooperation, this time having a written support.

A selection of papers from this volume will be considered for publication in special issues of Theoretical Computer Science.

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

- BWMC 2003: *Natural Computing* – volume 2, number 3, 2003, and *New Generation Computing* – volume 22, number 4, 2004;
- BWMC 2004: *Journal of Universal Computer Science* – volume 10, number 5, 2004, and *Soft Computing* – volume 9, number 5, 2005;
- BWMC 2005: *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006);
- BWMC 2006: *Theoretical Computer Science* – volume 372, numbers 2-3, 2007;

- BWMC 2007: *International Journal of Unconventional Computing* – volume 5, number 5, 2009;
- BWMC 2008: *Fundamenta Informaticae* – volume 87, number 1, 2008;
- BWMC 2009: *International Journal of Computers, Control and Communication* – volume 4, number 3, 2009;
- BWMC 2010: *Romanian Journal of Information Science and Technology* – volume 13, number 2, 2010;
- BWMC 2011: *International Journal of Natural Computing Research* – volume 2, numbers 2-3, 2011.
- BWMC 2012: *International Journal of Computer Mathematics* – volume 99, number 4, 2013.
- BWMC 2013: *Romanian Journal of Information Science and Technology*, vol. 17, nr. 1, 2014.
- BWMC 2014: *Fundamenta Informaticae*, volume 134, numbers 1-2, 2014.
- BWMC 2015: *Natural Computing* – volume 15, issue 4, 2016 (some papers from ACMC 2015 were also selected for this special issue).
- BWMC 2016: *Theoretical Computer Science* – in press (some papers from ACMC 2016 were also selected for this special issue).

Some of the papers elaborated during the 15th BWMC were submitted to various journals and 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. José Antonio Andreu-Guzmán, Universidad de Sevilla (Spain)
   `andreuguzman36@gmail.com`
2. Lucie Ciencialová, Silesian University (Opava, Czech Republic)
   `ciecilka@gmail.com`
3. Rudolf Freund, Technological University of Vienna (Austria)
   `rudifreund@gmx.at`
4. Carmen Graciani, Universidad de Sevilla (Spain)
   `cgdiaz@us.es`
5. Sergiu Ivanov, Université Paris-Est Créteil (France)
   `sivanov@colimite.fr`
6. Gábor Kolonits, Eötvös Loránd University (Hungary)
   `kolonits.gabor@gmail.com`
7. Alberto Leporati, University of Milan-Bicocca (Italy)
   `leporati@disco.unimib.it`
8. Francisco J. Macías-García, Universidad de Sevilla (Spain)
   `franmaciassevilla@gmail.com`

9. Luca Manzoni, University of Milan-Bicocca (Italy)
luca.manzoni@disco.unimib.it
10. Miguel A. Martínez-del-Amor, Universidad de Sevilla (Spain)
mdelamor@us.es
11. David Orellana-Martín, Universidad de Sevilla (Spain)
dorellana@us.es
12. Ignacio Prez-Hurtado, Universidad of Seville (Spain)
perezh@us.es
13. Mario de J. Pérez-Jiménez, Universidad de Sevilla (Spain) and Academia Europaea
marper@us.es
14. Antonio Enrico Porreca, University of MilanBicocca (Italy)
porreca@disco.unimib.it
15. Fernando Rendn-Segador, Universidad de Sevilla (Spain)
fernandojrs90@gmail.com
16. Agustín Riscos-Núñez, Universidad de Sevilla (Spain)
ariscosn@us.es
17. Daniel Rodríguez-Chavarría, Universidad de Sevilla (Spain)
danrodcha@gmail.com
18. Álvaro Romero-Jiménez, Universidad de Sevilla (Spain)
romero.alvaro@us.es
19. Eduardo Sánchez-Karhunen, Universidad de Sevilla (Spain)
sanchek@gmail.com
20. Jose María Sempere-Luna, Politechnical University of Valencia (Spain)
jsempere@dsic.upv.es
21. Luis Valencia-Cabrera, Universidad de Sevilla (Spain)
lvalencia@us.es
22. Gyorgy Vaszil, University of Debrecen (Hungary)
vaszil.gyorgy@inf.unideb.hu
23. Lian Ye, Chongqing University (China)
ylredleaf@cqu.edu.cn

As mentioned above, the meeting was organized by the Research Group on Natural Computing from Sevilla University (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 by Universidad de Sevilla, more precisely by Department of Computer Science and Artificial Intelligence and *VI Plan Propio, Vicerrectorado de Investigación de la Universidad de Sevilla.*

Carmen Graciani
Gheorghe Păun
Agustín Riscos-Núñez
Luis Valencia-Cabrera
(August 2017)

Contents

Unfair P Systems

Artiom Alhazov[1,2⋆], Rudolf Freund[3], and Sergiu Ivanov[4,5]

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

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

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

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

[5] TIMC-IMAG/DyCTiM, Faculty of Medicine of Grenoble
5 avenue du Grand Sablon, 38700, La Tronche, France
sergiu.ivanov@univ-grenoble-alpes.fr

Summary. We introduce a novel kind of P systems in which the application of rules in each step is controlled by a function on the applicable multisets of rules. Some examples are given to exhibit the power of this general concept. Moreover, for three well-known models of P systems we show how they can be simulated by P systems with a suitable fairness function.

1 Introduction

Membrane computing is a research field originally founded by Gheorghe Păun in 1998, see [5]. Membrane systems (also known as P systems) are a model of computing based on the abstract notion of a membrane and the rules associated to it which control the evolution of the objects inside. In many variants of P systems,

⋆ The work is supported by National Natural Science Foundation of China (61320106005 and 61033003) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

the objects are plain symbols from a finite alphabet, but P systems operating on more complex objects (e.g., strings, arrays) have been considered, too, e.g., see [2].

A comprehensive overview of different flavors of membrane systems and their expressive power is given in the handbook, see [6]. For a state of the art snapshot of the domain, we refer the reader to the P systems website [9] as well as to the bulletin series of the International Membrane Computing Society [8].

In this paper we introduce a novel kind of P systems in which the application of rules in each step is controlled by a function on the applicable multisets of rules, possibly also depending on the current configuration; we call this function the *fairness function*. In the standard variant, the fairness function will be used to choose those applicable multisets for which the fairness function yields the minimal value.

After recalling some preliminary notions and definitions in the next section, in Section 3 we will define our new model of *fair P systems* and give some examples to exhibit the power of this general concept. In Section 4, for three well-known models of P systems we will show how they can be simulated by P systems with a suitable fairness function. Future research topics finally are touched in Section 5.

2 Preliminaries

In this paper, the set of positive natural numbers $\{1, 2, \dots\}$ is denoted by $\mathbb{N}_+$, the set of natural numbers also containing 0, i.e., $\{0, 1, 2, \dots\}$, is denoted by $\mathbb{N}$. The set of integers denoted by $\mathbb{Z}$.

An *alphabet* V is a finite set. A (non-empty) *string* s over an alphabet V is defined as a finite ordered sequence of elements of V.

A *multiset* over V is any function $w : V \to \mathbb{N}$; $w(a)$ is the *multiplicity* of a in w. A multiset w is often represented by one of the strings containing exactly $w(a)$ copies of each symbol $a \in V$; the set of all these strings representing the multiset w will be denoted by $str(w)$. The set of all multisets over the alphabet V is denoted by V°. By abusing string notation, the empty multiset is denoted by λ.

The families of sets of Parikh vectors as well as of sets of natural numbers (multiset languages over one-symbol alphabets) obtained from a language family F are denoted by PsF and NF, respectively. The family of recursively enumerable string languages is denoted by RE.

For further introduction to the theory of formal languages and computability, we refer the reader to [6, 7].

2.1 (Hierarchical) P Systems

A *hierarchical P system* (P system, for short) is a construct

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

where O is the alphabet of objects, $T \subseteq O$ is the alphabet of terminal objects, μ is the membrane structure injectively labeled by the numbers from $\{1, \dots, n\}$

and usually given by a sequence of correctly nested brackets, w_i are the multisets giving the initial contents of each membrane i ($1 \leq i \leq n$), R_i is the finite set of rules associated with membrane i ($1 \leq i \leq n$), and h_i and h_o are the labels of the input and the output membranes, respectively ($1 \leq h_i \leq n$, $1 \leq h_o \leq n$).

In the present work, we will mostly consider the *generative case*, in which Π will be used as a multiset language-generating device. We therefore will systematically omit specifying the input membrane h_i.

Quite often the rules associated with membranes are multiset rewriting rules (or special cases of such rules). Multiset rewriting rules have the form $u \to v$, with $u \in O^o \setminus \{\lambda\}$ and $v \in O^o$. If $|u| = 1$, the rule $u \to v$ is called *non-cooperative*; otherwise it is called *cooperative*. Rules may additionally be allowed to send symbols to the neighboring membranes. In this case, for rules in R_i, $v \in O \times Tar_i$, where Tar_i contains the targets *out* (corresponding to sending the symbol to the parent membrane), *here* (indicating that the symbol should be kept in membrane i), and in_j (indicating that the symbol should be sent into the child membrane j of membrane i).

In P systems, rules are often applied in the maximally parallel way: in any derivation step, a non-extendable multiset of rules has to be applied. The rules are not allowed to consume the same instance of a symbol twice, which creates competition for objects and may lead to the P system choosing non-deterministically between the maximal collections of rules applicable in one step.

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

Example 1. For readability, we will often prefer a graphical representation of P systems; moreover, we will use labels to identify the rules. For example, the P system $\Pi_1 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1)$ with the rule set $R_1 = \{1 : a \to aa, 2 : a \to b\}$ may be depicted as in Figure 1.

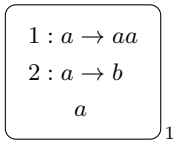


Fig. 1. The example P system Π_1

Due to maximal parallelism, at every step Π_1 may double some of the symbols a, while rewriting some other instances into b.

Note that, even though Π_1 might express the intention of generating the set of numbers of the powers of two, it will actually generate the whole of $\mathbb{N}_+$ (due to

the halting condition). Indeed, for any $n \in \mathbb{N}_+$, a^n can be generated in n steps by choosing to apply, in the first $n-1$ steps, $1 : a \to aa$ to exactly one instance of a and $a \to b$ to all the other instances, and by applying $2 : a \to b$ to every a in the last step (in fact, for $n > 1$, in each step except the last one, in which $2 : a \to b$ is applied twice, both rules are applied exactly once, as exactly two symbols a are present, whereas all other symbols are copies of b). □

While maximal parallelism and halting by inapplicability have been standard ingredients from the beginning, various other derivation modes and halting conditions have been considered for P systems, e.g., see [6].

2.2 Flattening

The folklore flattening construction (see [6] for several examples as well as [3] for a general construction) is quite often directly applicable to many variants of P systems. Hence, also for the systems considered in this paper we will not explicitly mention how results are obtained by flattening.

3 P Systems with a Fairness Function

In this section we consider variants of P systems using a so-called *fairness function* for choosing a multiset of rules out of the set of all multisets of rules applicable to a configuration.

3.1 The General Idea of a Fairness Function in P Systems

Take any (standard) variant of P systems and any (standard) derivation mode. The application of a multiset of rules in addition can be guided by a function computed based on specific features of the underlying configuration and of the multisets of rules applicable to this configuration. The choice of the multiset of rules to be applied then depends on the function values computed for all the applicable multisets of rules.

Therefore, in general we extend the model of a hierarchical P system to the model of a *hierarchical P system with fairness function* (*fair P system* for short)

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

where f is the fairness function defined for any configuration C of Π, the corresponding set $Appl_\delta(\Pi, C)$ of multisets of rules from Π applicable to C in the given derivation mode δ, and any multiset of rules $R \in Appl_\delta(\Pi, C)$. We then use the values $f(C, Appl_\delta(\Pi, C), R)$ for all $R \in Appl_\delta(\Pi, C)$ to choose a multiset $R' \in Appl_\delta(\Pi, C)$ of rules to be applied to the underlying configuration C. A standard option for choosing R' is to require it to yield the minimal value for the fairness function, i.e., we require $f(C, Appl_\delta(\Pi, C), R') \leq f(C, Appl_\delta(\Pi, C)), R)$

for all $R \in Appl_\delta(\Pi, C)$. As usually the derivation mode δ will be obvious from the context, we often shall omit it.

The fairness function may be independent from the underlying configuration, i.e., we may write $f(Appl(\Pi, C), R)$ only; in the simplest case, f is even independent from $Appl(\Pi, C)$, hence, in this case we only write $f(R)$.

Fair or Unfair

One may argue that it is fair to use rules in such a way that each rule should be applied if possible and, moreover, all rules should be applied in a somehow *balanced* way. Hence, a fairness function for applicable multisets should compute the best value for those multisets of rules fulfilling these guidelines.

On the other hand, we may choose the multiset of rules to be applied in such a way that it is the *unfairest* one. In this sense, let us consider the following *unfair example*.

Example 2. Consider the P system $\Pi_1 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1)$ with the rule set $R_1 = \{1 : a \to aa, 2 : a \to b\}$ as considered in Example 1 together with the fairness function f_2 defined as follows: if a rule is applied n times then it contributes to the function value of the fairness function f_2 for the multiset of rules with 2^{-n}. The total value for $f_2(R)$ for a multiset of rules R containing k copies of rule $1 : a \to aa$ and m copies of rule $2 : a \to b$ then is the sum $2^{-k} + 2^{-m}$. The resulting fair P system $\Pi_2 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1, f_2)$ is depicted in Figure 2; we observe that it can also be written as (Π_1, f_2).

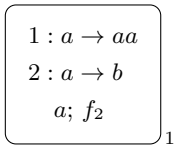


Fig. 2. The P system Π_2

In this fair P system (or in this case we might also call it maximally unfair) with one membrane working in the maximally parallel way, we again start with the axiom a and use the two rules $1 : a \to aa$ and $2 : a \to b$. If we apply only one of these rules to all m objects a, then the function value is 2^{-m} and is minimal compared to the function values computed for a mixed multiset of rules using both rules at least once.

Starting with the axiom a we use the rule $1 : a \to aa$ in the maximal way k times thus obtaining 2^k symbols a. Then in the last step, for all a we use the rule $2 : a \to b$ thus obtaining 2^k symbols b. We cannot mix the two rules in one of the derivation steps as only the clean use of exactly one of them yields the minimal value for the fairness function.

We observe that the effect is similar to that of controlling the application of rules by the well-known control mechanism called label selection, e.g., see [4], where either the rule with label 1 or the rule with label 2 has to be chosen. We will return to this model in Section 4.3. □

The following weird example shows that the fairness function should be chosen from a suitable class of (at least recursive) functions, as otherwise the whole computing power comes from the fairness function:

Example 3. Take the fair P system Π_3 with one membrane working in the maximally parallel way, starting with the axiom a and using the three rules $1: \ a \to aa$, $2: \ a \to a$, and $3: \ a \to b$, see Figure 3.

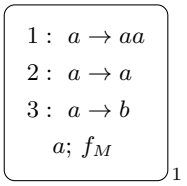


Fig. 3. The P system Π_3

Moreover, let $M \subset \mathbb{N}_+$, i.e., an arbitrary set of positive natural numbers. The fairness function f_M on multisets of rules over these three rules and a configuration containing m symbols a is defined as follows: For any multiset of rules R containing copies of the rules $1: \ a \to aa$, $2: \ a \to a$, and $3: \ a \to b$,

- $f(R) = 0$ if R only contains m copies of rule 3 and $m \in M$,
- $f(R) = 0$ if R only contains exactly one copy of rule 1 and the rest are copies of rule 2,
- $f(R) = 1$ for any other applicable multiset of rules.

Again the choice is made by applying only multisets of rules which yield the minimal value $f(R) = 0$. If we use rule $1: \ a \to aa$ once and rule $2: \ a \to a$ for the rest, this increases the number of symbols a in the skin membrane by one. Thus, in $m-1$ steps we get m symbols a. If m is in M, we now may use rule $3: \ a \to b$ for all symbols a, thus obtaining m symbols b, and the system halts. In that way, the system generates exactly $\{b^m \mid m \in M\}$.

To make this example a little bit less weird, we may only allow computable sets M. Still, the whole computing power is in the fairness function f_M alone, with f_M only depending on the multiset of rules. □

We now again return to Example 2 and illustrate how the same result can be obtained by using another fairness function in the standard *unfair* mode using the multsets of rules with minimal fairness value; on the other hand, we will also show what happens if we try to be *fair* and use the rules in a *balanced* way.

Example 4. Consider the P system $\Pi_1 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1)$ with the rule set $R_1 = \{1 : a \to aa, 2 : a \to b\}$ as considered in Example 1 together with the fairness function $f_4(R)$ for any multiset R of rules defined as follows: consider $f_4(R) = |str(R)|$, i.e., $f_4(R)$ is the number of different strings representing the multiset R. The resulting fair P system $\Pi_4 = (\Pi_1, f_4) = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1, f_4)$ is depicted in Figure 4.

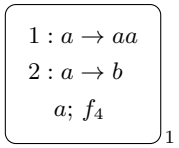


Fig. 4. The P system Π_4

With the standard selection of multisets of rules to be applied by choosing those with the minimal value of the fairness function, we obtain the same result for the set of multisets generated by Π_4, i.e., $\{a^{2^n} \mid n \in \mathbb{N}\}$, because only the pure multisets of rules R containing only copies of rule 1 or only copies of rule 2 yield $f(R) = 1$, whereas any mixed multiset of rules containing both rules at least once yields a bigger value.

On the other hand, if we try to be *fair* and use both rules in a balanced way, i.e., by choosing those multisets of rules yielding the maximum values of f_4, then the generated set is the singleton $\{b\}$, which can be generated in one step from the axiom a by using rule $2 : a \to b$. Any other derivation starting with using rule $1 : a \to aa$ will not yield any result due to running into an infinite computation without any chance to halt: as soon as aa has been generated, only once the rule $1 : a \to aa$ and once the rule $2 : a \to b$ can be used as only this combination of rules yields $f_4(\langle 1, 2\rangle) = |\{12, 21\}| = 2 > 1 = f_4(\langle 1, 1\rangle) = f_4(\langle 2, 2\rangle)$ (we here use the brackets $\langle, \rangle$ to describe a multiset). □

The problem with halting observed in the example above when using only non-cooperative rules seems to be an inherent one when using a *fair* (balanced) selection of multisets of rules. These variants may deserve further investigations in the future, but in this paper we will restrict ourselves to the standard (*maximally unfair*) selection of multisets of rules to be applied as in the previous examples.

4 First Results

In this section, we show three general results. The first one describes how priorities can be simulated by a suitable fairness function in P systems of any kind working in the sequential mode. The second one exhibits how P systems with energy control, see [1], can be simulated by suitable fair P systems for any arbitrary derivation

mode. Finally we show how P systems with rule label control, see [4], can be simulated by suitable fair P systems for any arbitrary derivation mode.

4.1 Simulating Priorities in the Sequential Derivation Mode

In the sequential derivation mode, exactly one rule is applied in every derivation step of the P system Π. Given a configuration C and the set of applicable rules $Appl(\Pi, C)$ not taking into account a given priority relation $<$ on the rules, we define the fairness function to yield 0 for each rule in $Appl(\Pi, C)$ for which no rule in $Appl(\Pi, C)$ with higher priority exists, and 1 otherwise. Thus, only a rule with highest priority can be applied. More formally, this result now is proved for any kind of P systems working in the sequential derivation mode:

Theorem 1. *Let $(\Pi, <)$ be a P system of any kind with the priority relation $<$ on its rules and working in the sequential derivation mode. Then there exists a fair P system (Π, f) with fairness function f simulating the computations in $(\Pi, <)$ selecting the multisets of rules with minimal values.*

Proof. First we observe that the main ingredient Π is exactly the same in both $(\Pi, <)$ and (Π, f), i.e., we only replace the priority relation $<$ by the fairness function f. As already outlined above, for any configuration C of Π we now define f for any rule r as follows (we point out that here the fairness function not only depends on $\{r\}$, but also on $Appl(\Pi, C)$):

- $f(Appl(\Pi, C), \{r\}) = 0$ if and only if there exists no rule $r' \in Appl(\Pi, C)$ such that $r < r'$, and
- $f(Appl(\Pi, C), \{r\}) = 1$ if and only if there exists a rule $r' \in Appl(\Pi, C)$ such that $r < r'$.

If we now define the task of f as choosing only those rules with minimal value, i.e., a rule r can be applied to configuration C if and only if $f(Appl(\Pi, C), \{r\}) = 0$, then we obtain the desired result. □

4.2 Simulating Energy Control

Recently we have considered P systems where a specific amount of energy is assigned to each rule, see [1]. There, only those multisets of rules are applied which use the minimal amount of energy. In a similar way the amount of energy coming up with a multiset of rules can be seen as the value of the fairness function. The minimal amount of energy then exactly corresponds with the minimal fairness.

In this paper, from the two variants of energy-controlled P systems we only consider the one where the energy is directly assigned to the rules. This variant of P systems is called a *rule energy-controlled* P system. The multisets or sets of rules to be applied to a given configuration must fulfill the condition of yielding the minimal amount of energy.

Formally, in a rule energy-controlled P system the rules are of the form (p, v) where p is a rule of a specific type like cooperative or non-cooperative and v is an integer energy value. The total energy value of a mutiset of rules can be defined in different ways, but in the following we will assume it to simply be the sum of energy values of the rules in the multiset and denote this function computing the energy value of a multiset of rules in this way by σ.

Theorem 2. *Let (Π, σ) be a rule energy-controlled P system working in any derivation mode, using any kind of rules and using the sum function σ for computing the energy value of a multiset of rules. Then there exists a fair P system (Π', f) with fairness function f simulating the computations in (Π, σ) with f selecting the multisets of rules with minimal values.*

Proof. By definition, in the rule energy-controlled P system (Π, σ) a multiset of rules can be applied to given configuration only if the application of σ yields the minimal value in $\mathbb{Z}$. The fair P system (Π', f) with fairness function f now is constructed from (Π, σ) by replacing any rule with energy (p, v) by the rule p itself, but on the other hand defining the fairness function f for a multiset of rules to take v as the value assigned to the rule p having been obtained from (p, v). By summing up these values for the whole multiset and selecting only those multisets of rules applicable to a given configuration in the given derivation mode which have minimal values, f fulfills the same task in (Π', f) as σ does in (Π, σ). Hence, in any derivation mode, (Π', f) simulates exactly step by step the derivations in (Π, σ), obviously yielding the same computation results. □

4.3 Simulating Label Selection

In P systems with label selection only rules belonging to one of the predefined subsets of rules can be applied to a given configuration, see [4].

For all the variants of P systems defined in Section 2, we may consider to label all the rules in the sets $R_1, \ldots, R_m$ in a one-to-one manner by labels from a set H and to take a set W containing subsets of H. Then a *P system with label selection* is a construct

$$\Pi^{ls} = (O, T, \mu, w_1, \ldots, w_n, R_1, \ldots R_n, h_i, h_o, H, W),$$

where $\Pi = (O, T, \mu, w_1, \ldots, w_n, R_1, \ldots R_n, h_i, h_o)$ is a P system as in Section 2, H is a set of labels for the rules in the sets $R_1, \ldots, R_m$, and $W \subseteq 2^H$. In any transition step in Π^{ls} we first select a set of labels $U \in W$ and then apply a non-empty multiset R of rules applicable in the given derivation mode restricted to rules with labels in U.

The following proof exhibits how the fairness function can also be used to capture the underlying derivation mode.

Theorem 3. *Let (Π, H, W) be a P system with label selection using any kind of rules in any kind of derivation mode. Then there exists a fair P system (Π', f) with fairness function f simulating the computations in (Π, H, W) with f selecting the multisets of rules with minimal values.*

Proof. By definition, in the P system (Π, H, W) with label selection a multiset of rules can be applied to given configuration only if all the rules have labels in a selected set of labels $U \in W$. We now consider the set of all multisets of rules applicable to a configuration C, denoted by $Appl_{asyn}(\Pi, C)$, as it corresponds to the asynchronous derivation mode (abbreviated $asyn$); from those we select all R which obey to the label selection criterion, i.e., there exists a $U \in W$ such that the labels of all rules in R belong to U, and then only take those which also fulfill the criteria of the given derivation mode restricted to rules with labels from U.

Hence we define (Π', f) by taking $\Pi' = \Pi$ and, for any derivation mode δ, f_δ for any multiset of rules $R \in Appl_{asyn}(\Pi, C)$ as follows:

- $f_\delta(C, Appl_{asyn}(\Pi, C), R) = 0$ if there exists a $U \in W$ such that the labels of all rules in R belong to U, and, moreover, $R \in Appl_\delta(\Pi_U, C)$, where Π_U is the restricted version of Π only containing rules with labels in U, as well as
- $f_\delta(C, Appl_{asyn}(\Pi, C), R) = 1$ otherwise.

According to our standard selection criterion, we choose only those multisets of rules where the fairness function yields the minimal value 0, i.e., those R such that there exists a $U \in W$ such that the labels of all rules in R belong to U and R is applicable according to the underlying derivation mode with rules restricted to those having a label in U, which exactly mimicks the way of choosing R in (Π, H, W). Therefore, in any derivation mode δ, (Π', f_δ) simulates exactly step by step the derivations in (Π, H, W), obviously yielding the same computation results. □

5 Conclusions and Future Research

In this article, we introduced and partially studied P systems with the application of rules in each step being controlled by a function on the applicable multisets of rules.

We have given several examples exhibiting the power of using suitable fairness functions. Moreover, we have shown how priorities can be simulated by a suitable fairness function in P systems of any kind working in the sequential mode as well as how P systems with energy control or label selection can be simulated by fair P systems with a suitable fairness function for any derivation mode.

Yet with all these examples and results we have just given a glimpse on what could be investigated in the future for P systems in connection with fairness functions:

- consider other variants of hierarchical P systems working in different derivation modes, e.g., also taking into consideration the set derivation modes;
- extend the notion of *fair* to tissue P systems, i.e., P systems on an arbitrary graph structure;
- extend the notion of fair to P systems with active membranes, there probably also controlling the division of membranes;
- investigate the effect of selecting the multiset of rules to be applied to a given configuration by other criteria than just taking those yielding the minimal values for the fairness function;
- consider other variants of fairness functions, either less powerful or taking into account other features of $Appl(\Pi, C)$ and/or the multiset of rules R;
- investigate the effect of selecting the multiset to be applied to a given configuration by requiring it to contain a balanced (really *fair*) amount of copies of each applicable rule;
- show similar simulation results with suitable fairness functions as in Section 4 for other control mechanisms used in the area of P systems;
- ...

References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Variants of energy-controlled P systems. In *Proceedings NIT 2016*, 2016.
2. Rudolf Freund. P systems working in the sequential mode on arrays and strings. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2004.
3. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.
4. Rudolf Freund, Marion Oswald, and Gheorghe Păun. Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundamenta Informaticae*, 136(1-2):59–84, 2015.
5. Gheorghe Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
6. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
7. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, 3 volumes*. Springer, New York, NY, USA, 1997.
8. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
9. The P Systems Website. `http://ppage.psystems.eu/`.

P Systems with Randomized Right-hand Sides of Rules

Artiom Alhazov[1,2⋆], Rudolf Freund[3], and Sergiu Ivanov[4,5]

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

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

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

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

[5] TIMC-IMAG/DyCTiM, Faculty of Medicine of Grenoble
5 avenue du Grand Sablon, 38700, La Tronche, France
sergiu.ivanov@univ-grenoble-alpes.fr

Summary. P systems are a model of hierarchically compartmentalized multiset rewriting. We introduce a novel kind of P systems in which rules are dynamically constructed in each step by non-deterministic pairing of left-hand and right-hand sides. We define three variants of right-hand side randomization and compare each of them with the power of conventional P systems. It turns out that all three variants enable non-cooperative P systems to generate exponential (and thus non-semi-linear) number languages. We also give a binary normal form for one of the variants of P systems with randomized rule right-hand sides. Finally, we also discuss extensions of the three variants to tissue P systems, i.e., P systems on an arbitrary graph structure.

⋆ The work is supported by National Natural Science Foundation of China (61320106005 and 61033003) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

1 Introduction

Membrane computing is a research field originally founded by Gheorghe Păun in 1998, see [13]. Membrane systems (also known as P systems) are a model of computing based on the abstract notion of a membrane. Formally, a membrane is treated as a container delimiting a region; a region may contain objects which are acted upon by the rewriting rules associated with the membranes. Quite often, the objects are plain symbols coming from a finite alphabet, but P systems operating on more complex objects (e.g., strings, arrays) are often considered, too, e.g., see [9].

A comprehensive overview of different flavors of membrane systems and their expressive power is given in the handbook which appeared in 2010, see [14]. For a state of the art snapshot of the domain, we refer the reader to the P systems website [17], as well as to the bulletin of the International Membrane Computing Society [16].

Dynamic evolution of the set of available rules has been considered from the very beginning of membrane computing. Already in 1999, generalized P systems were introduced in [8]; in these systems the membranes, alongside the objects, contain *operators* which act on these objects, while the P system itself acts on the operators, thereby modifying the transformations which will be carried out on the objects in the subsequent steps. Among further ideas on dynamic rules, one may list rule creation [4], activators [1], inhibiting/deinhibiting rules [7], and symport/antiport of rules [6]. One of the more recent developments in this direction are *polymorphic P systems* [2, 3, 12], in which rules are defined by pairs of membranes, whose contents may be modified by moving objects in or out.

We remark that the previous studies on dynamic rule sets either treated the rules as atomic entities (symport/antiport of rules, operators in generalized P systems), or allowed virtually unlimited possibilities of tampering with their shape (polymorphic P systems). In the present work, we propose a yet different approach which can be seen as an intermediate one.

In *P systems with randomized rule-right-hand sides* (or with randomized RHS, for short), the available left-hand sides and right-hand sides of rules are fixed, but the associations between them are *re-evaluated in every step*: a left-hand side may pick a right-hand side arbitrarily (randomly). In Section 3, we present three different formal definitions of this intuitive idea of randomized RHS:

1. rules *exchange* their RHS,
2. each rule randomly picks an RHS from a *common* collection of RHS, *shared* between the rules,
3. each rule randomly picks an RHS from a possible collection of *RHS associated with the rule itself*.

P systems with randomized RHS may have a real-world (possibly biological) application for representing systems in a hostile environment. The modifications such P systems effect on their rules may be used to represent perturbations caused

by the environment (mutations), somewhat in the spirit of faulty Turing machines (e.g., see [5]).

In this article, we will focus on the expressive power of P systems with randomized RHS, as well as on comparing them to the classical model with or without cooperative rules. One of the central conclusions of the present work is that non-cooperative P systems with randomized RHS can generate *exponential* number languages, thus (partially) surpassing the power of conventional P systems.

This paper is structured as follows. Section 2 recalls some preliminaries about multisets, strings, permutations, as well as conventional P systems. Section 3 defines the three variants of RHS randomization. Section 4 discusses the computational power of the three variants of P systems with randomized RHS. Section 5 shows a binary normal form for one of the variants of P systems with randomized RHS. Section6 discusses extensions of the three variants of RHS randomization to tissue P systems. Finally, Section 7 summarizes the results of the article and gives some directions for future work.

2 Preliminaries

In this paper, the set of positive natural numbers $\{1, 2, \dots\}$ is denoted by $\mathbb{N}^+$, the set of natural numbers also containing 0, i.e., $\{0, 1, 2, \dots\}$, is denoted by $\mathbb{N}$. Given $k \in \mathbb{N}^+$, we will call the set ${\mathbb{N}^+}_k = \{x \in \mathbb{N}^+ \mid 1 \leq x \leq k\}$ an *initial segment* of $\mathbb{N}^+$.

An *alphabet* V is a finite set. The families of recursively enumerable, context-free, linear, and regular languages, and of languages generated by tabled Lindenmayer systems are denoted by RE, CF, LIN, REG, and $ET0L$, respectively. The families of sets of Parikh vectors as well as of sets of natural numbers (multiset languages over one-symbol alphabets) obtained from a language family F are denoted by PsF and NF, respectively.

For further introduction to the theory of formal languages and computability, we refer the reader to [14, 15].

2.1 Linear Sets over $\mathbb{N}$

A *linear* set over $\mathbb{N}$ generated by a set of vectors $A = \{\mathbf{a}_i \mid 1 \leq i \leq d\} \subset_{fin} \mathbb{N}^n$ (here $A \subset_{fin} B$ indicates that A is a finite subset of B) and an offset $\mathbf{a}_0 \in \mathbb{N}^n$ is defined as follows:

$$\langle A, \mathbf{a}_0 \rangle_{\mathbb{N}} = \left\{ \mathbf{a}_0 + \sum\nolimits_{i=1}^{d} k_i \mathbf{a}_i \;\middle|\; k_i \in \mathbb{N},\ 1 \leq i \leq d \right\}.$$

If the offset $\mathbf{a}_0$ is the zero vector $\mathbf{0}$, we call the corresponding linear set *homogeneous*; we also use the short notation $\langle A \rangle_{\mathbb{N}} = \langle A, \mathbf{0} \rangle_{\mathbb{N}}$.

We use the notation ${\mathbb{N}^n}LIN_{\mathbb{N}} = \{\langle A, \mathbf{a}_0 \rangle_{\mathbb{N}} \mid A \subset_{fin} \mathbb{N}^n,\ \mathbf{a}_0 \in \mathbb{N}^n\}$, to refer to the class of all linear sets of n-dimensional vectors over $\mathbb{N}$. Semi-linear sets are

defined as finite unions of linear sets. We use the notation $\mathbb{N}^n SLIN_{\mathbb{N}}$ to refer to the classes of semi-linear sets of n-dimensional vectors. In case no restriction is imposed on the dimension, n is replaced by $*$. We may omit n if $n = 1$. A finite union of linear sets which only differ in the starting vectors is called *uniform* semilinear:

$$\mathbb{N}^n SLIN_{\mathbb{N}}^U = \left\{ \bigcup_{\mathbf{b} \in B} \langle A, \mathbf{b} \rangle_{\mathbb{N}} \mid A \subset_{fin} \mathbb{N}^n,\ B \subset_{fin} \mathbb{N}^n \right\}$$

Let us denote such a set by $\langle A, B \rangle_{\mathbb{N}}$.

Note that a uniform semilinear set $\langle A, B \rangle_{\mathbb{N}}$ can be seen as a pairwise sum of the finite set B and the homogeneous linear set $\langle A \rangle_{\mathbb{N}}$:

$$\langle A, B \rangle_{\mathbb{N}} = \{ \mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \langle A \rangle_{\mathbb{N}}, \mathbf{b} \in B \}.$$

This observation immediately yields the conclusion that the sum of two uniform semilinear sets $\langle A_1, B_1 \rangle_{\mathbb{N}}$ and $\langle A_2, B_2 \rangle_{\mathbb{N}}$ is uniform semilinear as well and can be computed in the following way:

$$\langle A_1, B_1 \rangle_{\mathbb{N}} + \langle A_2, B_2 \rangle_{\mathbb{N}} = \{ \mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \langle A_1 \cup A_2 \rangle_{\mathbb{N}}, \mathbf{b} \in B_1 + B_2 \}.$$

As is folklore,

$$PsCF = PsLIN = PsREG = \mathbb{N}^* SLIN_{\mathbb{N}}.$$

2.2 Multisets

A *multiset* over V is any function $w : V \to \mathbb{N}$; $w(a)$ is the *multiplicity* of a in w. A multiset w is often represented by one of the strings containing exactly $w(a)$ copies of each symbol $a \in V$. The set of all multisets over the alphabet V is denoted by V°. By abusing string notation, the empty multiset is denoted by λ. The *projection* (restriction) of w over a sub-alphabet $V' \subseteq V$ is the multiset $w|_{V'}$ defined as follows:

$$w|_{V'}(a) = \begin{cases} w(a), & a \in V'; \\ 0, & a \in V \setminus V'. \end{cases}$$

Example 1. The string aab can represent the multiset $w : \{a, b\} \to \mathbb{N}$ with $w(a) = 2$ and $w(b) = 1$. The projection $w|_{\{a\}} = w'$ is defined as $w'(a) = w(a) = 2$ and $w'(b) = 0$.

We will (ab)use the symbol $\in$ to denote the relation "is a member of" for multisets. Therefore, for a multiset w, $a \in w$ will stand for $w(a) > 0$.

2.3 Strings and Permutations

A (non-empty) *string* s over an alphabet V traditionally is defined as a finite ordered sequence of elements of V. Equivalently, we can define a string of length

k as a function assigning symbols to positions: $s : {\mathbb{N}^+}_k \to V$. Thus, the string $s = aab$ can be equivalently defined as the function $s : {\mathbb{N}^+}_3 \to \{a, b\}$ with $s(1) = a$, $s(2) = a$, and $s(3) = b$. We will use the traditional notation $|s|$ to refer to the length of the string s (i.e., the size k of the initial segment ${\mathbb{N}^+}_k$ it is defined on). In addition, the size of the empty string λ is 0.

A string $s : {\mathbb{N}^+}_k \to V$ is not necessarily surjective (there may be symbols from V that do not appear in s). We will use the notation $set(s)$ to refer to the set of symbols appearing in s (the image of s):

$$set(s) = \left\{ a \in V \mid a = s(i) \text{ for some } i \in {\mathbb{N}^+}_{|s|} \right\}.$$

Given a string $s : {\mathbb{N}^+}_k \to V$, a *prefix* of length $k' \leq k$ of s is the restriction of s to ${\mathbb{N}^+}_{k'} \subseteq {\mathbb{N}^+}_k$. For example, aa is a prefix of length 2 of the string aab. We will use the notation $\text{pref}_{k'}(s)$ to denote the prefix of length k' of s.

Given a finite set A, a *permutation* of A is any bijection $\rho : A \to A$. Given a permutation $\sigma : {\mathbb{N}^+}_k \to {\mathbb{N}^+}_k$ and a string $s : {\mathbb{N}^+}_k \to V$ of length k, *applying* σ *to* s is defined as $\sigma(s) = s \circ \sigma$ (where $\circ$ is the function composition operator).

Example 2. Following the widespread tradition, we will write permutations in Cauchy's two-line notation. The permutation σ_{rev} of ${\mathbb{N}^+}_3$ which "reverses the order" of the numbers, can be written as follows:

$$\sigma_{rev} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}.$$

Applying σ_{rev} to a string reverses it:

$$\sigma_{rev}(aab) = baa.$$

Any finite set B trivially can be represented by one of the strings listing all of its elements exactly once. All such strings are equivalent modulo permutations. Given a fixed enumeration $B = \{b_1, \ldots, b_n\}$, we define the *canonical string representation* of B to be the string $\delta(B) = b_1 \ldots b_n$.

2.4 Rule Sides

We consider arbitrary labeled multiset rules $r : u \to v$ over an alphabet V, where r is the rule label we attach for convenience, and u and v are strings over V representing multisets. As usual, the application of such a rule means replacing the multiset represented by u by the multiset represented by v.

For a given rule $r : u \to v$, we define the left-hand-side and the right-hand-side functions as follows:

$$lhs(u \to v) = lhs(r) = (u),$$
$$rhs(u \to v) = rhs(r) = (v).$$

Using the brackets (and), for a given string w, the notation (w) is used to describe the multiset represented by w. As usual, we will extend the notations

for these functions lhs and rhs lifted to sets of rules: given a set of rules R, $lhs(R) = \{lhs(r) \mid r \in R\}$ and $rhs(R) = \{rhs(r) \mid r \in R\}$. Furthermore, for any *string* (finite ordered sequence) of rules $\rho : {\mathbb{N}^+}_k \to R$ we define the strings of left-hand sides $lhs(\rho) = lhs \circ \rho$ and of right-hand sides $rhs(\rho) = rhs \circ \rho$.

Example 3. Take $R = \{r_1 : aa \to ab, r_2 : cc \to cd\}$ and consider the string of rules $\rho = r_1 r_1 r_2$. Then $lhs(\rho) = (aa)(aa)(cc)$ and $rhs(\rho) = (ab)(ab)(cd)$. Thus, $lhs(\rho)$ and $rhs(\rho)$ can be considered as *strings of multisets*.

We will (ab)use the symbol $\to$ for *combining* two strings of multisets $\alpha, \beta : {\mathbb{N}^+}_k \to V^\circ$ of the same length k. The *string* $\alpha \to \beta$ will be defined as follows, for any $i \in {\mathbb{N}^+}_k$:

$$(\alpha \to \beta)(i) = \alpha(i) \to \beta(i).$$

Example 4. Consider the following two strings of multisets: $\alpha = (aa)(aa)(cc)$ and $\beta = (ab)(ab)(cd)$. $\alpha \to \beta$ is simply the string of rules that can be obtained by taking the multisets from α as left-hand sides and β as right-hand sides, in the given order: $\alpha \to \beta = (aa) \to (ab)(aa) \to (ab)(cc) \to (cd)$ (which exactly corresponds with ρ from Example 3).

2.5 (Hierarchical) P Systems

A (hierarchical) *P system* is a construct

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

where O is the alphabet of objects, $T \subseteq O$ is the alphabet of terminal objects, μ is the membrane structure injectively labeled by the numbers from $\{1, \ldots, n\}$ and usually given by a sequence of correctly nested brackets, w_i are the multisets giving the initial contents of each membrane i ($1 \leq i \leq n$), R_i is the finite set of rules associated with membrane i ($1 \leq i \leq n$), and h_i and h_o are the labels of the input and the output membranes, respectively ($1 \leq h_i \leq n$, $1 \leq h_o \leq n$).

In the present work, we will mostly consider the *generative case*, in which Π will be used as a multiset language-generating device. We therefore will systematically omit specifying the input membrane h_i.

Quite often the rules associated with membranes are multiset rewriting rules (or special cases of such rules). Multiset rewriting rules have the form $u \to v$, with $u \in O^o \setminus \{\lambda\}$ and $v \in O^o$. If $|u| = 1$, the rule $u \to v$ is called *non-cooperative*; otherwise it is called *cooperative*. Rules may additionally be allowed to send symbols to the neighboring membranes. In this case, for rules in R_i, $v \in O \times Tar_i$, where Tar_i contains the targets *out* (corresponding to sending the symbol to the parent membrane), *here* (indicating that the symbol should be kept in membrane i), and in_h (indicating that the symbol should be sent into the child membrane h of membrane i). Note that all variants of the function rhs, as well as the operator $\to$ from the previous section can be naturally extended to rules having right-hand sides with target indications (from $O \times Tar_i$).

In P systems, rules are often applied in the maximally parallel way: in any derivation step, a non-extendable multiset of rules has to be applied. The rules are not allowed to consume the same instance of a symbol twice, which creates competition for objects and may lead to the P system choosing non-deterministically between the maximal collections of rules applicable in one step.

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

Example 5. For readability, we will often prefer a graphical representation of P systems. For example, the P system $\Pi_1 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R, 1)$ with the rule set $R = \{a \to aa, a \to b\}$ may be depicted as in Figure 1.

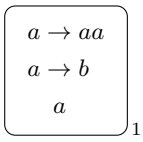


Fig. 1. The example P system Π_1

Due to maximal parallelism, at every step Π_1 may double some of the symbols a, while rewriting some other instances into b.

Note that, even though Π_1 might express the intention of generating the set of numbers of the powers of two, it will actually generate the whole of $\mathbb{N}^+$ (due to halting). Indeed, for any $n \in \mathbb{N}^+$, a^n can be generated in n steps by choosing to apply, in the first $n-1$ steps, $a \to aa$ to exactly one instance of a and $a \to b$ to all the other instances, and by applying $a \to b$ to every a in the last step (in fact, for $n > 1$, in each step except the last one, in which $a \to b$ is applied twice, both rules are applied exactly once, as exactly two symbols a are present, whereas all other symbols are copies of b).

While maximal parallelism and halting by inapplicability are staple ingredients, various other derivation modes and halting conditions have been considered for P systems, e.g., see [14].

We will use the notation $OP_n(coo)$ to denote the family of P systems with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

3 P Systems with Randomized RHS

In this section we consider three different variants of defining P systems with randomized RHS. We immediately point out that, despite the common intuitive background, the details of the resulting semantics vary quite a lot.

3.1 Variant 1: Random RHS Exchange

In this variant of P systems, rules randomly exchange right-hand sides at the beginning of every evolution step. This variant was the first to be conceived and is the closest to the classical definition.

A *P system with random RHS exchange* is a construct

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

where the components of the tuple are defined as in the classical model (Section 2.5).

As different from conventional P systems, Π does not apply the rules from R_i directly. Instead, for each membrane $1 \leq i \leq n$, we take the canonical representation of R_i, i.e., $\delta(R_i)$, and non-deterministically (randomly) choose a permutation $\sigma : {\mathbb{N}^+}_{|R_i|} \to {\mathbb{N}^+}_{|R_i|}$ to compute the canonical representation of R_i^σ from $\delta(R_i)$ as follows:

$$\delta(R_i^\sigma) = lhs(\delta(R_i)) \to \sigma(rhs(\delta(R_i))).$$

We now extract the set of rules $R_i^\sigma = set(\delta(R_i^\sigma))$ described by the string $\delta(R_i^\sigma)$ as constructed above. Π will then apply the rules from R_i^σ according to the usual maximally parallel semantics in membrane i.

In other words, Π *non-deterministically permutes* the right-hand sides of rules in each membrane i, and then applies the obtained rules according to the maximally parallel semantics.

Note that we first have to transform the set R_i into its canonical string representation $\delta(R_i)$ in order to be able to obtain a correct representation of the $|R_i|$ rules and from that a correct representation of the $|R_i|$ rules in R_i^σ, even if the number of different left-hand sides and/or different right-hand sides of rules does not equal $|R_i|$.

Example 6. Consider the P system $\Pi_2 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R, 1)$ with the rule set $R = \{a \to aa, c \to b\}$. Π_2 is graphically represented in Figure 2.

The number language generated by Π_2 (the set of numbers of instances of b that may appear in the skin after Π_2 has halted) is exactly $\{2^n \mid n \in \mathbb{N}^+\}$. Indeed, while Π_2 applies the identity permutation on the right-hand sides, $a \to aa$ will double the number of symbols a, while the rule $c \to b$ will never be applicable. When Π_2 exchanges the right-hand sides of the rules, the rule $a \to b$ will rewrite every symbol a into a symbol b. After this has happened, no rule will ever be applicable any more and Π_2 will halt with 2^n symbols b in the skin, where $n + 1$ is the number of computation steps taken.

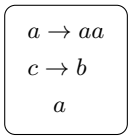


Fig. 2. The P system Π_2 with random RHS exchange generating the number language $\{2^n \mid n \in \mathbb{N}\}$.

We will use the notation

$$OP_n(rhsExchange, coo)$$

to denote the family of P systems with random RHS exchange, with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

3.2 Variant 2: Randomized Pools of RHS

In this variant of P systems, every membrane has some fixed left-hand sides and a *pool* of available right-hand sides to build rules from. An RHS from the pool can only be used once.

A *P system with randomized pools of RHS* is a construct

$$\Pi = (O, T, \mu, w_1, \ldots, w_n, H_1, \ldots H_n, h_o),$$

where H_i defines the left- and right-hand sides available in membrane i and the other components of the tuple are defined as in the classical model (Section 2.5).

For $1 \leq i \leq n$, $H_i = (l_i, r_i)$ is a pair of strings of multisets over O. The string r_i may contain target indications (i.e., be a string of multisets over $O \times Tar_i$). The strings l_i and r_i are not necessarily of the same length. The length of the shortest of the two strings l_i and r_i is denoted by

$$k_i = \min(|l_i|, |r_i|).$$

At the beginning of every computation step in Π, for every membrane i, we construct the set of rules it will apply in the following way:

1. non-deterministically choose two (random) permutations

$$\sigma_l : \mathbb{N}^+{}_{|l_i|} \to \mathbb{N}^+{}_{|l_i|}, \quad \sigma_r : \mathbb{N}^+{}_{|r_i|} \to \mathbb{N}^+{}_{|r_i|};$$

2. take the first k_i elements out of $\sigma_l(l_i)$ and $\sigma_r(r_i)$:

$$l'_i = \mathrm{pref}_{k_i}(\sigma_l(l_i)), \quad r'_i = \mathrm{pref}_{k_i}(\sigma_r(r_i));$$

3. construct the set of rules R_i to be applied in membrane i by combining the left- and right-hand sides from l'_i and r'_i:

$$R_i = set(l'_i \to r'_i).$$

In step (3), we combine the strings l'_i and r'_i using the operator $\to$ defined in Subsection 2.4 and then apply the operator *set* to obtain the corresponding set of rules from the string representation.

After having constructed the set R_i for each membrane i, Π will proceed to applying the obtained rules according to the usual maximally parallel semantics.

When computing the strings l'_i and r'_i, we apply *two* different permutations σ_l and σ_r to l_i and r_i, in order to ensure fairness for the participation of left-hand and right-hand sides when $|l_i| \neq |r_i|$. For example, if we only permuted r_i in the case in which $|l_i| > |r_i|$, the left-hand sides located at positions $k > |r_i|$ in l_i would never be used.

We do not explicitly prohibit repetitions in l_i or in r_i, but we avoid repeated rules by constructing R_i using the *set* function.

Example 7. Consider the following P system with randomized pools of RHS: $\Pi_3 = (\{a, b\}, \{b\}, [\,]_1, a, H, 1)$, with $H = \big((a), (aa)(b)\big)$; (a) stands for the multiset containing an instance of a, while $(aa)(b)$ is the string denoting the two multisets (aa) and (b). The graphical representation of Π_3 is given in Figure 3.

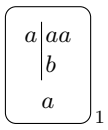


Fig. 3. The P system Π_3 with randomized pools of RHS generating the number language $\{2^n \mid n \in \mathbb{N}\}$.

The pair $H = (l, r)$ of strings of multisets is represented by listing the multisets of l and r in two columns and by drawing a vertical line between the two columns.

Π_3 follows exactly the same pattern as Π_2 from Example 6: while the identity permutation is applied to r, Π_3 keeps doubling the symbols a in the skin. Once the multisets (aa) and (b) are permuted in r, and thus the rule $a \to b$ is formed, all symbols a are rewritten into symbols b in one step and Π_3 must halt. Note that randomly taking the right-hand sides from a given pool avoids having the extra dummy rule $c \to b$ in Π_2.

We will use the notation

$$OP_n(rhsPools, coo)$$

to denote the family of P systems with randomized pools of RHS, with at most n membranes, with cooperative rules. To denote the family of such P systems with

non-cooperative rules, we replace *coo* by *ncoo*. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

3.3 Variant 3: Individual Randomized RHS

In this variant of P systems, each rule is constructed from a left-hand side and a set of possible right-hand sides.

A *P system with individual randomized RHS* is a construct

$$\Pi = (O, T, \mu, w_1, \ldots, w_n, P_1, \ldots P_n, h_o),$$

where P_i is the set of *productions* associated with the membrane i and the other components of the tuple are defined as in the classical model (Section 2.5).

A production is a pair $u \to R$, where $u \in O^\circ$ is the left-hand side and $R \subseteq O^\circ$ is a finite set of right-hand sides. The right-hand sides in R may have target indications, i.e., for a production in membrane i, we may consider $R \subseteq (O \times Tar_i)^\circ$. At the beginning of each computation step, for every membrane i, for each production $u \to R \in R_i$, Π will non-deterministically (randomly) pick a right-hand side v from R and will construct the rule $u \to v$ (this happens once per production). Π will then apply the rules thus constructed according to the maximally parallel semantics.

Example 8. Generating the language of the powers of two is the easiest compared with Variants 1 and 2. Indeed, consider the P system with individual randomized RHS $\Pi_4 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, P, 1)$ with only one production: $P = \{a \to \{aa, b\})\}$. Its graphical representation is given in Figure 4.

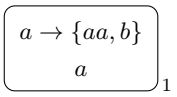


Fig. 4. The P system Π_4 with individual randomized RHS generating the number language $\{2^n \mid n \in \mathbb{N}\}$.

Π_4 works exactly like Π_2 and Π_3 from Examples 6 and 7: it doubles the number of symbols a and halts by rewriting them to b in the last step.

We will use the notation

$$OP_n(rndRhs, coo)$$

to denote the family of P systems with individual randomized RHS, with at most n membranes, with cooperative rules. To denote the family of such P systems with *non-cooperative* rules, we replace *coo* by *ncoo*. To denote the family of languages

of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N.

We will sometimes want to set an upper bound k on the number of right-hand sides per production. To refer to the family of P systems with individual randomized RHS with such an upper bound, we will replace *rndRhs* by $rndRhs^k$ in the notation above.

3.4 Halting with Randomized RHS

The conventional (total) halting condition for P systems can be naturally lifted to randomized RHS: a P system Π with randomized RHS (Variant 1, 2, or 3) halts on a configuration C if, however it permutes rule right-hand sides in Variant 1, or however it builds rules out of the available rule sides in Variants 2 and 3, no rule can be applied in C, in any membrane.

Note that, for Variants 1 and 3, the permutations chosen do *not* affect the applicability of rules, because applicability only depends on left-hand sides, which are always the same in any membrane. The situation is different for Variant 2, because the number of available left-hand sides in a membrane of Π may be bigger than the number of available right-hand sides. Therefore, if Π is a P system with randomized pools of RHS, the way rule sides are permuted may affect the number of rules applicable in a given configuration. This is why, for Π to halt on C, we require no rule to be applicable for any permutation.

In this paper, we will mainly consider P systems with randomized pools of RHS in which, in every membrane, there are at least as many right-hand sides as there are left-hand sides. To refer to P systems with this restriction, we will use the notation $rhsPools'$. In these systems, the problem with the applicability of rules as described above can be avoided.

3.5 Equivalence Between Variants 1 and 2

Before discussing the computational power of the P systems with randomized RHS in general, we will briefly point out a strong relationship between P systems with random RHS exchange and P systems with randomized pools of RHS, *with* the restriction that every membrane contains at least as many right-hand sides as it has left-hand sides, i.e., for P systems with randomized RHS of type $rhsPools'$.

Theorem 1. *For any $k \in \{coo, ncoo\}$, the following holds:*

$$PsOP_n(rhsExchange, k) = PsOP_n(rhsPools', k).$$

Proof. Any membrane with random RHS exchange trivially can be transformed into a membrane with randomized pools of RHS by listing the left-hand sides of the rules in the pool of LHS and the right-hand sides of the rules in the pool of RHS.

Conversely, consider a membrane i with randomized pools of RHS, with the string l_i of LHS and the string r_i of RHS, $|l_i| \leq |r_i|$. We can transform it into a membrane with random RHS exchange as follows. For every LHS u from l_i, pick (and remove) an RHS v from r_i, and construct the rule $u \to v$. According to our supposition, we will exhaust the LHS before (or at the same time as) the RHS. For every RHS v' which is left, we add a new (dummy) symbol z' to the alphabet, and add the rule $z' \to v'$. Since the symbol z' is new and does not appear in any RHS, it will never be produced and the rule $z' \to v'$ will essentially serve as a stash for the RHS v'. □

3.6 Flattening

The folklore flattening construction (see [14] for several examples as well as [10] for a general construction) is quite directly applicable to P systems with individual randomized RHS.

Proposition 1 (flattening). *For any $k \in \{coo, ncoo\}$, the following is true:*

$$PsOP_1(rndRhs, k) = PsOP_n(rndRhs, k).$$

Proof (sketch). Since in the case of individual randomized RHS, randomization has per rule granularity (whereas in the other two variants randomization occurs at the level of membranes), we can simulate multiple membranes by attaching membrane labels to symbols. For example, a production $ab \to \{cd, f\}$ in membrane h becomes $a_h b_h \to \{c_h d_h, f_h\}$, while the send-in production $a \to \{(b, in_i), (b, in_j)\}$ becomes $a_h \to \{b_i, b_j\}$. □

On the other hand, for Variants 1 and 2 similar results cannot be proved in such a way, a situation which happens very seldom in the area of P systems, especially in the case of variants of the standard model. Yet intuitively, it is easy to understand why this happens, as in both Variants 1 and 2 the right-hand sides in just one membrane can randomly be chosen for any left-hand side, whereas different membranes can separate the possible combinations of left-hand sides and right-hand sides of rules. A formal proof showing that flattening is impossible for the types *rhsExchange* and *rhsPools*$'$ will be given in the succeeding section by constructing a suitable example.

4 Computational Power of Randomized RHS

In this section, we look into the computational power of the three different versions of P systems with randomized right-hand sides. We first shortly consider the case of cooperative rules and then focus on the case of non-cooperative rules.

4.1 Cooperative Rules

The following result concerning the relationship between P systems with individual randomized RHS and conventional P systems holds for both cooperative and non-cooperative rules:

Proposition 2. *For any* $n \in \mathbb{N}^+$ *and* $\alpha \in \{ncoo, coo\}$, $PsOP_n(rndRhs, \alpha) \supseteq PsOP_n(\alpha)$.

Proof. Any conventional P system can be trivially seen as a P system with individual randomized RHS in which every production has exactly one right-hand side. □

Now, the computational completeness of *cooperative* P systems trivially implies the computational completeness of P systems with individual randomized RHS.

Corollary 1. *For any* $n \in \mathbb{N}^+$, $PsOP_n(rndRhs, coo) = PsRE$.

4.2 Non-cooperative Rules

First we mention an upper bound for the families $PsOP_n(\rho, ncoo)$, for any variant $\rho \in \{rhsExchange, rhsPools', rndRhs\}$:

Proposition 3. *For any* $n \in \mathbb{N}^+$ *and* $\rho \in \{rhsExchange, rhsPools', rndRhs\}$,

$$PsOP_n(\rho, ncoo) \subseteq PsET0L.$$

Proof. No matter how the rule sets are constructed in the three different variants, we always get a finite set of different sets of rules—*tables*—corresponding to tables in $ET0L$-systems, which can also mimic the contents of different membranes in the usual way by using symbols marked with the corresponding membrane label. □

Next we show one of the central results of this paper: randomized rule right-hand sides allow for generating *non-semilinear languages* already in the non-cooperative case.

Theorem 2. *The following is true for* $\rho \in \{rhsExchange, rhsPools', rndRhs\}$*:*

$$\{2^m \mid m \in \mathbb{N}\} \in NOP_n(\rho, ncoo) \setminus NOP_n(ncoo).$$

Proof. The statement follows (for $n \geq 1$) from the constructions given in Examples 6, 7, and 8 and from the well-known fact that non-cooperative P systems operating under the total halting condition cannot generate non-semilinear number languages (for example, see [14]). □

This result is somewhat surprising at a first glance, but becomes less so when one remarks that the constructions from all three examples only effectively use *one rule* to do the multiplication, which is non-deterministically changed to a "halting" rule. Since there is only one rule acting at any time, randomized right-hand sides allow for clearly delimiting different *derivation phases*.

It turns out that this approach of synchronization by randomization can be exploited to generate even more complex non-semilinear languages.

Theorem 3. *Given a fixed subset of natural factors* $\{f_1, \ldots, f_k\} \subseteq \mathbb{N}$*, the following is true for any* $\rho \in \{rhsExchange, rhsPools', rndRhs\}$*:*

$$L = \{f_1^{n_1} \cdot \ldots \cdot f_k^{n_k} \mid n_1, \ldots, n_k \in \mathbb{N}\} \in NOP_1(\rho, ncoo).$$

Proof. First consider the P system with randomized pools of RHS $\Pi_5 = (\{a, b\}, \{b\}, [_1\ ]_1, a, H, 1)$ with $H = (l, r)$, $l = (a)$ and $r = \left(a^{f_1}\right) \ldots \left(a^{f_k}\right) \left(b\right)$. This P system is graphically represented in Figure 5.

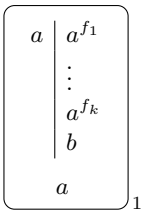


Fig. 5. The P system Π_5 with randomized pools of RHS generating the number language $\{f_1^{n_1} \cdot \ldots \cdot f_k^{n_k} \mid n_1, \ldots, n_k \in \mathbb{N}\}$.

Similarly to the P systems from Examples 6, 7, and 8, Π_5 halts by choosing to pick the right-hand side b and constructing the rule $a \to b$. If Π_5 picks a different right-hand side, it will multiply the contents of the skin membrane (membrane 1) by one of the factors f_i, $1 \leq i \leq k$. This proves that $L \in NOP_1(rhsPools', ncoo)$, and, according to Theorem 1, $L \in NOP_1(rhsExchange, ncoo)$ as well: take the P system with the rules $\{a \to a^{f_1}, z_2 \to a^{f_2}, \ldots, z_k \to a^{f_k}, z_{k+1} \to b\}$ (the rules with z_j in their left-hand sides are dummy rules).

To show that $L \in NOP_1(rndRhs, ncoo)$, just construct a P system with the only production $a \to \{a^{f_1}, \ldots, a^{f_k}, b\}$. □

Therefore, randomizing the right-hand sides of rules in non-cooperative P systems allows for generating non-semilinear languages which cannot be generated without randomization. A natural question to ask is whether randomizing the RHS leads to a *strict increase* in the computational power. The answer is trivially positive for P systems with individual randomized RHS (Variant 3).

Proposition 4. *For any* $n \in \mathbb{N}^+$*,* $PsOP_n(rndRhs, ncoo) \supsetneq PsOP_n(ncoo)$.

Proof. The inclusion follows from Proposition 2, as any conventional P system can be trivially seen as a P system with individual randomized RHS in which every production has exactly one right-hand side. Theorem 3 proves the strictness of the inclusion. □

On the other hand, the other two variants of randomizing right-hand sides—random RHS exchange (Variant 1) and randomized pools of RHS (Variant 2)—actually *prevent* one-membrane P systems with non-cooperative rules from generating some semilinear languages, which result also shows that flattening is not possible for these two variants.

In what follows, we will use the expression "only one rule is applied" to refer to the fact that only one given rule $u \to v$ is applied in a certain configuration, possibly in multiple copies. Dually, by saying "at least two rules are applied", we mean that at least two different rules, $u \to v$ and $u' \to v'$, are applied, possibly in multiple copies each.

Theorem 4. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following holds:*

$$L_{ab} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\} \notin PsOP_1(\rho, ncoo).$$

Proof. Consider a P system Π with randomized RHS of the variant given by ρ and with non-cooperative rules. We immediately remark that no left-hand side in Π may be a or b, because in this case Π will never be able to halt with its only (skin) membrane containing either the multiset a^n or b^n. Furthermore, any RHS of Π contains combinations of symbols a, b, or LHS symbols. Indeed, if an RHS contained a symbol not belonging to these three classes, instances of this symbol would pollute the halting configuration. Finally, Π contains no RHS v such that $a \in v$ and $b \in v$. If Π did contain such an RHS, then any computation could be hijacked to produce a mixture of symbols a and b.

With these remarks in mind, the statement of the theorem follows from the contradicting Lemmas 1 and 2, which are shown immediately after this proof. □

Lemma 1. *Take a $\Pi \in OP_1(\rho, ncoo)$, $\rho \in \{rhsExchange, rhsPools'\}$, such that it generates the number language $Ps(\Pi) = L_{ab}$. Then it must have a computation in which more than one rule is applied (two different left-hand sides are employed) in at least one step.*

Proof. Suppose that Π applies exactly one rule in every step of every computation. We make the following two remarks:

1. Since the words in L_{ab} are of unbounded length, Π must have an LHS t and an RHS v such that $t \in v$, otherwise all computations of Π would have one step and would only produce words of bounded length.
2. Every such RHS v must contain at most one kind of LHS, i.e., if t_1 and t_2 are two LHS of Π then $t_1 \in v$ and $t_2 \in v$ implies $t_1 = t_2$. If this were not the case, after using v, Π would *have* to apply two different rules (assuming that Π has at least as many RHS as LHS).

According to these observations, as well as to those from the proof of Theorem 4, any RHS v of Π is the of the form $v = \alpha\beta$, where $\alpha \in \{a^k, b^k \mid k \in \mathbb{N}\}$, $\beta \in \{t^k \mid k \in \mathbb{N}\}$, and t is an LHS of Π. Note that both α and β may be empty. According to observation (1), Π must have at least an RHS for which $\beta \neq \lambda$ and there exists such an RHS which must be applied an unbounded number of times.

In what follows, we will separately treat the cases in which Π contains or does not contain mixed RHS, i.e., RHS for which both $\alpha \neq \lambda$ and $\beta \neq \lambda$.

No mixed RHS:

Suppose that any RHS of Π which contains a left-hand side is of the form t_2^k. Then, according to our previous observations on the possible forms of the RHS of Π, all RHS containing a are of the form a^i and all RHS containing b are of the form b^j. According to the remarks from the proof of Theorem 4, a and b must not be LHS of Π. Therefore, in any computation of Π, all of a's and b's are produced in the last step. But then, the number of terminal symbols Π produces in a computation can be calculated as a product of the sizes of the RHS of the rules it has applied, which implies that there exists such a $p \in \mathbb{N}$ such that $a^p \notin Ps(\Pi)$ and therefore $Ps(\Pi) \neq L_{ab}$. (p may be picked to be the smallest prime number greater than the length of the longest RHS of Π.)

Mixed RHS:

It follows from the previous paragraph that, in order to generate the number language L_{ab}, Π should contain and apply at least one RHS of the form $a^i t_1^{k_1}$ and at least one RHS of the form $b^j t_2^{k_2}$. Take a computation C of Π producing a and applying the rule $t \to a^i t_1^{k_1}$ at a certain step. Instead of this rule, apply $t \to b^j t_2^{k_2}$, and, in the following step, the rule $t_2 \to a^i t_1^{k_1}$. (We can do so because Π is allowed to pick any permutation of RHS.) Now, Π may continue applying the same rules as in C and eventually halt with a configuration containing *both* a and b. This implies that $Ps(\Pi) \neq L_{ab}$.

It follows from our reasoning that, if Π applies exactly one rule in any step of any computation, it cannot produce L_{ab}, which proves the lemma. □

Lemma 2. *Take a $\Pi \in OP_1(\rho, ncoo)$, $\rho \in \{rhsExchange, rhsPools'\}$, such that it generates the number language $Ps(\Pi) = L_{ab}$. Then, in every computation of Π, exactly one rule is applied (one left-hand side is employed) in every step.*

Proof. Suppose that, in every computation of Π, there exists a step at which at least two different rules are applied. This immediately implies that Π has no RHS of the form a^i or b^j, for $i, j \geq 0$. Indeed, consider a computation producing the multiset a^n and a step in it at which more than one rule is applied. Then Π can replace one of the RHS introduced into the system at this step by b^j and thus end up with a mix of a's and b's in the halting configuration. Therefore, all RHS of Π containing a have the form $a^i v_a$ and all RHS containing b have the form $b^j v_b$,

where v_a and v_b are non-empty multisets which only contain LHS symbols (which are neither a nor b).

Now, consider a computation C_a of Π halting on the multiset a^n, and take the *last* step s_a at which at least two different rules are applied. We will consider three different cases, based on whether a and an LHS t appear in the configurations of C_a *after* step s_a.

Both a and t are present:

Suppose both a and an LHS t are present at step $s_a + 1$ in computation C_a. Then t is the only LHS present, because, by our hypothesis, only one rule is applied (maybe in multiple instances) at step $s_a + 1$. In this case, replace the rule applied at step $s_a + 1$ in C_a by $t \to b^j v_b$, where $b^j v_b$ is a right-hand side of Π used in a computation C_b producing b's. From step $s_a + 2$ on in the modified computation, just apply the same rules as applied to the symbols of v_b (and to those derived from v_b) in C_b. The modified computation will reach a halting configuration containing a mix of a's and b's.

Only a is present:

Suppose only a is present at step $s_a + 1$ in computation C_a. Then all of the RHS used at step s_a are λ, because Π has no RHS of the form a^i. Then, replace one of these empty RHS by $b^j v_b$, where $b^j v_b$ is a right-hand side of Π used in a computation C_b producing b's. As before, just apply the same rules as in C_b in the modified computation to get a mix of a's and b's in the halting configuration.

No symbols a are present:

Suppose now that there are no instances of a present at step $s_a + 1$ in computation C_a. Recall that Π has no RHS of the form a^i. Since we suppose that s_a is the last step at which at least two different rules are applied, this means that, in order to produce any a's in C_a, Π must have and use an RHS of the form $a^i t^k$. This RHS contains (multiple copies of) exactly one kind of LHS symbol: t.

Consider a computation C_b halting on the multiset b^n. We pick n sufficiently big to ensure that C_b uses at least two RHS containing b: $b^j v_b$ and $b^{j'} v'_b$ (possibly the same). Without losing generality, we may suppose that these two RHS are either used at the same step in C_b or that $b^{j'} v'_b$ is used at a later step than $b^j v_b$. Then, replace $b^{j'} v'_b$ by $a^i t^k$, pick one of the LHS symbols $t' \in v'_b$ and apply the same rules to t (and to the symbols derived from t) in the modified derivation as were applied to t' (and to the symbols derived from t') in C_b. The modified derivation will therefore contain a mix of a's and b's in the halting configuration.

It follows from our reasoning that, if in any derivation of Π there is a step at which at least two different rules are applied, then $Ps(\Pi) \neq L_{ab}$, which proves the lemma. □

The previous two lemmas are contradicting each other, which means that there exist no one-membrane P systems with random RHS exchange or with random pools of RHS which generate the union language $L_{ab} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\}$ (this is the statement of Theorem 4). Together with Theorem 3, this leads us to the curious conclusion that one-membrane non-cooperative P systems with random RHS exchange or with randomized pools of RHS are *incomparable* in power to the conventional P systems.

Corollary 2. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following two statements are true:*

$$PsOP_1(\rho, ncoo) \setminus PsOP_1(ncoo) \neq \emptyset, \tag{1}$$
$$PsOP_1(ncoo) \setminus PsOP_1(\rho, ncoo) \neq \emptyset. \tag{2}$$

Proof. Statement (1) follows from Theorem 3. Statement (2) follows from Theorem 4. □

Theorem 4 also allows us to draw a negative conclusion as to the computational completeness of one-membrane non-cooperative P systems with random RHS exchange (Variant 1) and non-cooperative P systems with randomized pools of RHS (Variant 2).

Corollary 3. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following is true:*

$$PsOP_1(\rho, ncoo) \subsetneq PsRE.$$

It turns out that allowing multiple membranes strictly increases the expressive power of Variants 1 and 2 and allows for easily generating *all semilinear* languages, as shown by the following theorem.

Theorem 5. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following holds:*

$$\mathbb{N}^* SLIN_{\mathbb{N}} \in PsOP_*(\rho, ncoo).$$

Proof. Consider the following semilinear language of d-dimensional vectors $L = \bigcup_{1 \leq i \leq n} \langle A_i, \mathbf{b}_i \rangle_{\mathbb{N}}$, where $A_i \subset_{fin} \mathbb{N}^d$ and $\mathbf{b}_i \in \mathbb{N}^d$. We construct the corresponding P system with randomised pools of RHS:

$$\Pi_6 = \left(O, T, [\,[\,]_2 \ldots [\,]_{n+1}]_1, w_0, \lambda, \ldots, \lambda, H_1, \ldots H_{n+1}, 1\right),$$

with the alphabet and the initial contents of the skin defined as follows:

- $O = \{a_1, \ldots, a_d, t\}$ contains a symbol per each dimension of the vectors, plus the special symbol t,
- $T = \{a_1, \ldots, a_d\}$ contains exactly one symbol per dimension of vectors,
- $w_0 = t$.

The pools of LHS and RHS $H_1 = (l_1, r_1)$ associated with the skin membrane 1 of Π_6 are:

$$l_1 = (t), \quad r_1 = \big(u_1\,(t, in_2)\big) \ldots \big(u_n\,(t, in_{n+1})\big),$$

where the multiset u_i corresponds to the offset $\mathbf{b}_i$: $Ps(u_i) = \mathbf{b}_i$, $1 \le i \le n$. Finally, the pools of rule sides $H_{i+1} = (l_{i+1}, r_{i+1})$ associated with inner membrane $i+1$ are defined as follows:

$$l_{i+1} = (t), \quad r_{i+1} = \big(t\,(v_{i1}, out)\big) \ldots \big(t\,(v_{ik_i}, out)\big)\,(\lambda),$$

where the multisets v_{ij}, $1 \le j \le k_i$, correspond to the vectors of the set $A_i = \{\mathbf{a}_{i1}, \ldots, \mathbf{a}_{ik_i}\}$: $Ps(v_{ij}) = \mathbf{a}_{ij}$, $1 \le j \le k_i$. By abuse of notation, we write (w, out) to mean that every symbol instance in w gets the target indication out. Π_6 is graphically represented in Figure 6.

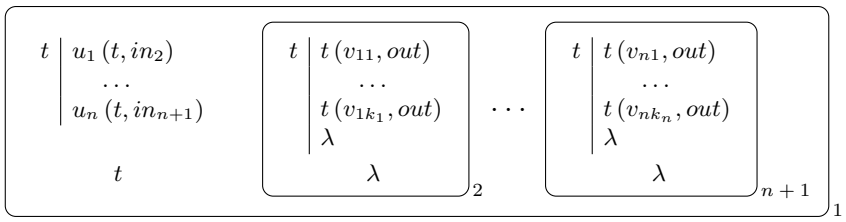


Fig. 6. The P system Π_6 with randomized pools of RHS generating the semilinear language $L = \bigcup_{1 \le i \le n} \langle A_i, \mathbf{b}_i \rangle_{\mathbb{N}}$.

Π_6 starts by non-deterministically building one of the rules $t \to u_i\,(t, in_{i+1})$ in the skin membrane. An application of this rule adds the multiset corresponding to the offset $\mathbf{b}_i$ to the skin membrane and puts t into inner membrane $i+1$. In the following steps only rules in membrane $i+1$ may become applicable. In this membrane, Π_6 may build rules of the form $t \to t\,(v_{ij}, out)$, $1 \le j \le k_i$, which will sustain t while also sending the multiset v_{ij} corresponding to the vector $\mathbf{a}_{ij} \in A_i$ out into the skin. Alternatively, Π_6 may choose to build the rule $t \to \lambda$, an application of which will erase t and halt the system. In such a computation, Π_6 generates the multiset language corresponding to $\langle A_i, \mathbf{b}_i \rangle_{\mathbb{N}}$. Since Π_6 can choose to send t into any one of its inner membranes in the first step and since the computations of said membranes cannot interfere, we conclude that $Ps(\Pi_6) = L$.

To complete the proof, we evoke Theorem 1 to show that there exists a P system with random RHS exchange (Variant 1) generating the same language L.

This theorem allows us to draw a definitive conclusion about the impossibility of flattening for non-cooperative Variants 1 and 2, in contrast to Proposition 1 showing the opposite result for Variant 3.

Corollary 4. *For $\rho \in \{rhsExchange, rhsPools'\}$ and any $k \ge 2$, the following holds:*

$$PsOP_1(\rho, ncoo) \subsetneq PsOP_k(\rho, ncoo).$$

We conclude this section with two more observations regarding the computational power of the Variants 1 and 2. We have seen that, with a single membrane and without cooperation, such P systems cannot generate all semilinear languages; yet it turns out they can generate all *uniform semilinear* languages.

Theorem 6. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following is true:*

$$\mathbb{N}^* SLIN_{\mathbb{N}}^{U} \subseteq PsOP_1(\rho, ncoo).$$

Proof. Consider two finite sets of d-dimensional vectors $A, B \subset_{fin} \mathbb{N}^d$, $A = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $B = \{\mathbf{y}_1 \ldots, \mathbf{y}_m\}$, and the uniform semilinear set $\langle A, B\rangle_{\mathbb{N}}$. We will now construct the P system $\Pi = (O, T, [\ ]_1, w_0, H, 1)$ with pools of randomized RHS in the following way:

- $O = \{a_1, \ldots, a_d, t\}$ contains a symbol per each dimension of the vectors, plus the special symbol t,
- $T = \{a_1, \ldots, a_d\}$ contains exactly one symbol per dimension of vectors,
- $w_0 = t$,
- $H = (l, r)$, with $l = (t)$ and $r = (w'_1 t) \ldots (w'_n t)\,(v'_1) \ldots (v'_m)$, such that $Ps(w'_i) = \mathbf{x}_i$, $1 \leq i \leq n$, and $Ps(v'_j) = \mathbf{y}_j$, $1 \leq j \leq m$.

In every step, Π either chooses one of the RHS $(w'_i t)$ which will enable it to reuse the left-hand side symbol t in the following step, or it constructs a rule of the form $t \to v'_j$, which erases the only instance of t and halts the system. Thus, Π performs arbitrary additions of vectors $\mathbf{x}_i \in A$ and then, in the last step of the computation, introduces one of the initial offsets $\mathbf{y}_j \in B$. Therefore, $Ps(\Pi) = \langle A, B\rangle_{\mathbb{N}}$. The fact that we can construct such a P system Π for any uniform semilinear set proves the statement of the theorem. □

Even though one-membranenon-cooperative P systems with random RHS exchange and with randomized pools of RHS cannot generate all unions of linear languages (Theorem 4), they can still generate some limited unions of exponential languages.

Theorem 7. *For $\rho \in \{rhsExchange, rhsPools'\}$, the following is true:*

$$L'_{ab} = \left\{a^{2^n} \mid n \in \mathbb{N}\right\} \cup \left\{b^{2^n} \mid n \in \mathbb{N}\right\} \in PsOP_1(\rho, ncoo).$$

Proof. A P system Π_7 generating the language L'_{ab} can be constructed as follows: $\Pi_7 = (\{a, b, t\}, \{a, b\}, [\ ]_1, t, H, 1)$, where $H = (l, r)$, $l = (t)$ and $r = (tt)(a)(b)$. A graphical representation of Π_7 is given in Figure 7.

Π_7 works by sequentially multiplying the number of symbols t by 2, until it decides to rewrite every instance of t to a or every instance of t to b. Therefore, $Ps(\Pi_7) = L'_{ab}$. According to Proposition 1, there also exists a P system with random RHS exchange generating L'_{ab}, which completes the proof. □

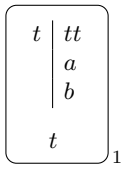


Fig. 7. The P system Π_7 with randomized pools of RHS generating the union language $L'_{ab} = \left\{a^{2^n} \mid n \in \mathbb{N}\right\} \cup \left\{b^{2^n} \mid n \in \mathbb{N}\right\}$

The construction from the previous proof can be clearly extended to any number of distinct terminal symbols and to any function of the number of steps $f(n)$ given by a product of exponentials (like in Theorem 3). That is, one can construct a P systems with random RHS exchange or with randomized pools of RHS generating the union language $\left\{a_i^{f(n)} \;\middle|\; n \in \mathbb{N}, 1 \le i \le m\right\}$, for some fixed number m. Note, however, that we cannot use the same approach to generate unions of two different exponential functions. We conjecture that generating such unions is entirely impossible with Variants 1 and 2 of randomized RHS.

5 Variant 3: A Binary Normal Form

In this section we present a binary normal form for P systems with individual randomized RHS: we prove that, for any such P system, there exists an equivalent one in which every production has at most two right-hand sides.

We now introduce a (rather common) construction: symbols with finite timers attached to them. Given an alphabet O, we define the following two functions:

$$timers_o(t, O) = \bigcup_{i=1}^{t} \{\langle a, i\rangle \mid a \in O\},$$
$$timers_r(t) = \{\langle a, i\rangle \to \langle a, i-1\rangle \mid 2 \le i \le t\} \cup \{\langle a, 1\rangle \to a \mid a \in O\}.$$

Informally, $timers_o(t, O)$ attaches a t-valued timer to every symbol in O, while $timers_r(t)$ contains the rules making this timer work.

We also define the following function setting a timer to the value $t > 0$ for each symbol in a given string $a_1 \dots a_n$:

$$wait(t, a_1 \dots a_n) = \langle a_1, t\rangle \dots \langle a_n, t\rangle.$$

For $t = 0$, $wait$ is defined to be the identity function: $wait(0, a_1 \dots a_n) = a_1 \dots a_n$.

We can now show that, for any P system with individual randomized RHS there exists an equivalent one having at most two RHS per production.

Theorem 8 (normal form). *For any $\Pi \in OP_n(rndRhs, k)$, $k \in \{coo, ncoo\}$, there exists a $\Pi' \in OP_n(rndRhs^2, k)$ such that $Ps(\Pi') = Ps(\Pi)$.*

Proof. Consider the following P system with individual randomized RHS $\Pi = (O, T, \mu, w_1, \ldots, w_n, P_1, \ldots P_n, h_o)$ that has at least one production with more than two RHS. We will construct another P system with individual randomized RHS $\Pi' = (O', T, \mu, w_1, \ldots, w_n, P'_1, \ldots P'_n, h_o)$ such that $Ps(\Pi') = Ps(\Pi)$. The new alphabet will be defined as

$$O' = O \cup timers_o(t, O) \cup \{p_1, \ldots, p_t \mid p \in V_p\},$$

where $t + 2$ is the number of right-hand sides in the productions of Π having the most of them, and V_p is an alphabet containing a symbol for each of the individual productions of Π. (If there are two identical productions in Π which belong to two different membranes, V_p will contain one different symbol for each of these two productions.)

For every membrane $1 \leq i \leq n$, the new set of productions P'_i is constructed by applying the following procedure to every production $p \in P_i$:

- If p has the form $u \to \{v\}$, we add the production $u \to \{wait(t, v)\}$ to P'_i.
- If p has the form $u \to \{v_1, v_2\}$, we add $u \to \{wait(t, v_1), wait(t, v_2)\}$ to P'_i.
- If p has the form $u \to \{v_1, \ldots, v_k\}$, with $k \geq 3$, we add the following productions to P_i:

$$\begin{aligned} &\big\{u \to \{wait(t, v_1), p_1\}\big\} \\ \cup\ &\big\{p_j \to \{wait(t - j, v_{j+1}), p_{j+1}\} \mid 1 \leq j < k - 2\big\} \\ \cup\ &\big\{p_{k-2} \to \{wait(t - k + 2, v_{k-1}), wait(t - k + 2, v_k)\}\big\}. \end{aligned}$$

These productions are graphically represented in Figure 8, in which arrows go from LHS to the associated RHS.

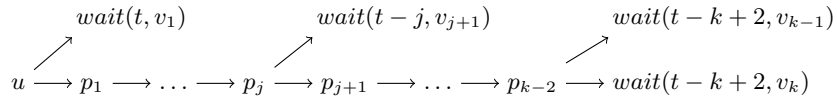


Fig. 8. Timers allow sequential choice between any number of right-hand sides.

Finally we add the rules from $timers_r(t)$, treated as one-RHS production, to every P'_i.

Instead of directly choosing between the right hand-sides of a production $p : u \to \{v_1, \ldots, v_k\}$ in one step, Π' chooses between v_1 and delaying the choice to the next step, by producing p_1. This choice between settling on an RHS or continuing the enumeration in the next step may be kept on until $k - 2$ RHS have been discarded. If p_{k-2} is reached, Π' must choose one of the two remaining RHS.

Thus, Π' evolves in "macro-steps", each consisting of exactly t steps. In the first step of a "macro-step", Π' acts on the symbols from O, producing some symbols with timers and delaying some of the choices by producing symbols p_j. All symbols with timers wait exactly until the t-th step of the "macro-step" to

turn into the corresponding clean versions from O. Since $t+2$ is the number of RHS in the biggest production of Π, Π' has the time to enumerate all of the RHS of this production.

Since every delayed choice of Π' is uniquely identified by a production-specific symbol p_j, and since only the productions from $timers_r(t)$ can act upon the symbols with timers in Π', the simulations of two different productions of Π cannot interfere. This concludes the proof of the normal form. □

6 Tissue P Systems with Randomized Right-hand Sides of Rules

We now extend the idea of randomized right-hand sides of rules to tissue P systems, where the underlying graph structure is an arbitrary graph structure and not a rooted tree as in the case of hierarchical P systems. Moreover, we also might allow every cell to interact with the environment in case the underlying variant of tissue P system allows/requires that, yet in the following we will assume one of the n cells to figure as the environment, thus being the only cell in which some elementary objects may appear infinitely often

Following the general notation as described for networks of cells in [11], we define a tissue P system as follows:

A *tissue P system* is a construct

$$\Pi = (n, O, T, w_1, \ldots, w_n, R, h_i, h_o),$$

where n is the number of cell, labeled by 1 to n, O is the alphabet of objects, $T \subseteq O$ is the alphabet of terminal objects, w_i are the multisets giving the initial contents of each cell i ($1 \leq i \leq n$), R is the finite set of rules, and h_i and h_o are the labels of the input and the output cells, respectively ($1 \leq h_i \leq n$, $1 \leq h_o \leq n$). If e is the label of the environment, then w_e may contain an infinite part. The rules in R are of the form

$$(u_1, \ldots, u_n) \to (v_1, \ldots, v_n)$$

interpreted as follows: the multisets u_i are replaced by the multisets v_i, $1 \leq i \leq n$. Such a rule can also be written as follows:

$$\prod_{i=1}^{n} (i, u_i) \to \prod_{i=1}^{n} (i, v_i)$$

Special ingredients can be added to the rules, for example promoters P_i (which have to be present in cell i) and/or inhibitors Q_i (which must not be present in cell i), with P_i and Q_i being finite sets of multisets from O; then a rule

$$((u_1, \ldots, u_n) \to (v_1, \ldots, v_n); (P_1, \ldots, P_n), (Q_1, \ldots, Q_n))$$

is applicable to a configuration if and only if cell i contains all elements of P_i and no element from Q_i, $1 \leq i \leq n$.

Now let m rules be given as

$$\prod_{i=1}^{n}(i,u_i^{(k)}) \to \prod_{i=1}^{n}(i,v_i^{(k)}), \quad 1 \leq k \leq m.$$

According to the general definition of tissue P systems as given above, the rules are not assigned to specific cells but to the whole tissue P system (although assigning rules to cells is another interesting variant to be investigated in the future). For the rules we now have several possibilities to interpret the randomization of the right-hand sides of rules:

Variant A This variant in the strictest way resembles the way randomization was defined for hierarchical P systems:
For a rule $\prod_{i=1}^{n}(i,u_i^{(k)}) \to \prod_{i=1}^{n}(i,v_i^{(k)})$, we simply take $\prod_{i=1}^{n}(i,v_i^{(k)})$ as the right-hand side of the rule and then define Variants 1, 2, and 3 as for hierarchical P systems.

Variant B For the Variants 1 and 2, the right-hand sides $\prod_{i=1}^{n}(i,v_i^{(k)})$ of the m rules are separated into the elements $v_1^{(k)}$ to $v_n^{(k)}$ and the elements $v_i^{(k)}$ for each cell i, $1 \leq i \leq n$, are randomized independently, i.e., we take the multisets

$$M_i = \langle v_i^{(k)} \mid 1 \leq k \leq m \rangle$$

as starting points for randomization and for constructing the rules by taking out one element from M_i for each i, $1 \leq i \leq n$, to construct the right-hand side of a rule.

Variant C As a special variant of Variant B, for randomization in Variants 1 and 2 we only take those $v_i^{(k)}$ for which $v_i^{(k)} \neq \lambda$, i.e., we now instead take the multisets

$$M_i' = \langle v_i^{(k)} \mid v_i^{(k)} \neq \lambda, 1 \leq k \leq m \rangle = \langle x \in M_i \mid x \neq \lambda \rangle.$$

Moreover, we may consider two subvariants how to construct the new right-hand sides of rules:

Variant C.1 If M_i' is empty, then we cannot construct any randomized rule.

Variant C.2 If M_i' is empty, then we take (i,λ) for every constructed randomized rule.

We observe that for Variant 3, i.e., for individual randomized RHS, we only consider Variant A. Therefore, for all three Variants 1 to 3 we will use the notation

$$OtP_n(\alpha, X)$$

to denote the family of tissue P systems with at most n cells using rules of type X with α denoting the type of randomization according to Variants 1 to 3. To denote the family of languages of multisets generated by these P systems, we prepend Ps to the notation, and to denote the family of the generated number languages, we prepend N. For the Variants 1 and 2, we may also add an additional parameter $\beta \in \{B, C.1, C.2\}$ (to indicate how to deal with empty $v_i^{(k)}$) thus obtaining the notations $OtP_n(\alpha, \beta, X)$ etc.

6.1 Equivalence Between Variants 1 and 2 for Variant A

For randomized pools of RHS, again we consider the restriction that there are at least as many right-hand sides as it has left-hand sides for the rules to be constructed, i.e., the type $rhsPools'$. Then again we obtain the equivalence between tissue P systems with random RHS exchange and tissue P systems with randomized pools of RHS of type $rhsPools'$. The proof follows the same lines as the proof of Theorem 1, now taking into account that we only have to consider the whole system (or, if rules are assigned to cells, we simply replace *membrane* by *cell*).

Proposition 5. *For any $n \in \mathbb{N}^+$ and $X \in \{coo, ncoo\}$, the following holds:*

$$PsOtP_n(rhsExchange, X) = PsOtP_n(rhsPools', X).$$

7 Conclusions and Open Problems

In this article, we introduced and partially studied P systems with randomized rule right-hand sides. This is a model of P systems with dynamic rules, in which the matching between left-hand and right-hand sides is non-deterministically changed during the evolution. In each step, such P systems first construct the rules from the available rule sides and then apply them, in a maximally parallel way.

We defined three different randomization semantics: random RHS exchange (Variant 1), randomized pools of RHS (Variant 2), and individual randomized RHS (Variant 3). We studied the computational power of the three variants and showed that Variant 3 is quite different in power from Variants 1 and 2. Indeed, P systems with individual randomized RHS (Variant 3) appear as a strict extension of conventional P systems, while random RHS exchange (Variant 1) and randomized pools of RHS (Variant 2) seem to increase the power when only one LHS is used, but to decrease the power when more LHS are present. Finally, we gave a binary normal form for P systems with individual randomized RHS (Variant 3).

7.1 Open Questions

The present work leaves open quite a number of open questions. We list the ones appearing important to us, in no particular order.

Full power of Variants 1 and 2:

Are cooperative, multi-membrane P systems with random RHS exchange (Variant 1) or with randomized pools of RHS (Variant 2) computationally complete? If not, what would be the upper bound on their power? In this article, we showed that applying these two randomization semantics to the non-cooperative, one-membrane case, yields a family of multiset languages incomparable with the family of semi-linear vector sets. How much more can be achieved with cooperativity?

We conjecture that, even with LHS containing more than one symbol, Variants 1 and 2 will *not* be computationally complete. However, we expect that considering systems with multiple membranes may actually bring a substantial boost in computational power, because, in both Variants 1 and 2, randomization happens over each single membrane, meaning that one might use a rich membrane structure to finely control its effects.

Compare the variants:

How do the three variants of RHS randomization compare among one another when applied to non-cooperative rules? We saw that, in all three cases, exponential number languages can be generated. We also saw that individual randomized RHS (Variant 3) produce a strict superset of the semi-linear languages (Proposition 4). Does it imply that Variant 3 is strictly more powerful than Variants 1 and 2? We conjecture a positive answer to this question.

Excess of LHS:

In the case of P systems with randomized pools of RHS (Variant 2), what is the consequence of having *more LHS* available in a membrane than there are RHS? The results in this paper concern a "restricted" version of Variant 2, in which we require that LHS are never in excess. How strong is this restriction? Our conjecture is that allowing an excess of LHS does not increase the computational power.

Applications to vulnerable systems:

As noted in the introduction to the present work, randomized RHS can be seen as a representation of systems mutating in a toxic environment. However, we did not give any concrete examples. It would be interesting to look up any such concrete cases and to evaluate the relevance of this unconventional modeling approach.

7.2 Further Variants

Forbidding identical rules:

In any of the three variants, it may happen that identical rules are constructed, in any membrane. In the previous chapters, in this case this rule was simply taken into the set of rules. Yet we could also forbid such a situation to happen and in such a case completely abandon the whole rule set. Another solution can be to take out all rules having been constructed more than once from the constructed rule set.

The situation of getting identical rules can easily be avoided by avoiding identical RHS: the right-hand sides of rules can be made different by adding suitable powers of a dummy symbol d, which does not count for the final result (i.e., d is no terminal symbol). As d also does not appear on the left-hand side of a rule, the computational power of any of the P systems variant considered in this paper will not be changed by this changing of the set of RHS available for constructing the set of rules.

Identical RHS in Variant 3:

In P systems with individual randomized RHS the computational power mainly arises from the possibility to specify different sets of RHS for the left-hand sides of rules. What happens if the set R of RHS must be the same for all left-hand sides?

References

1. Artiom Alhazov. A note on P systems with activators. In Gheorghe Păun, Agustín Riscos-Núñez, Alvaro Romero-Jiménez, and Fernando Sancho-Caparrini, editors, *Second Brainstorming Week on Membrane Computing, Sevilla, Spain, February 2-7 2004*, pages 16–19, 2004.
2. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Marion Oswald. Observations on P systems with states. In Marian Gheorghe, Ion Petre, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Multidisciplinary Creativity. Hommage to Gheorghe Păun on His 65th Birthday*. Spandugino, 2015.
3. Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic P systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2011.
4. Fernando Arroyo, Angel V. Baranda, Juan Castellanos, and Gheorghe Păun. Membrane computing: The power of (rule) creation. *Journal of Universal Computer Science*, 8:369–381, 2002.
5. Ilir Çapuni and Péter Gács. A Turing machine resisting isolated bursts of faults. *CoRR*, abs/1203.1335, 2012.
6. Matteo Cavaliere and Daniela Genova. P systems with symport/antiport of rules. In Gheorghe Păun, Agustín Riscos-Núñez, Alvaro Romero-Jiménez, and Fernando Sancho-Caparrini, editors, *Second Brainstorming Week on Membrane Computing, Sevilla, Spain, February 2–7 2004*, pages 102–116, 2004.
7. Matteo Cavaliere, Mihai Ionescu, and Tseren-Onolt Ishdorj. Inhibiting/de-inhibiting rules in P systems. In *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, Italy, June 2004*, pages 174–183, 2004.
8. Rudolf Freund. Generalized P-Systems. In Gabriel Ciobanu and Gheorghe Păun, editors, *Fundamentals of Computation Theory, 12th International Symposium, FCT '99, Iaşi, Romania, August 30–September 3,1999, Proceedings*, volume 1684 of *Lecture Notes in Computer Science*, pages 281–292. Springer, 1999.
9. Rudolf Freund. P systems working in the sequential mode on arrays and strings. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2004.
10. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.

11. Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) p systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing: 8th International Workshop, WMC 2007 Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers*, pages 271–284. Springer, 2007.
12. Sergiu Ivanov. Polymorphic P systems with non-cooperative rules and no ingredients. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2014.
13. Gheorghe Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
14. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
15. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, 3 volumes*. Springer, New York, NY, USA, 1997.
16. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
17. The P Systems Website. `http://ppage.psystems.eu/`.

Time-freeness and Clock-freeness and Related Concepts in P Systems ⋆

Artiom Alhazov[1,2⋆⋆], Rudolf Freund[3], Sergiu Ivanov[4,5], Linqiang Pan[2,6], and Bosheng Song[2]

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

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

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

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

[5] TIMC-IMAG/DyCTiM, Faculty of Medicine of Grenoble,
5 avenue du Grand Sablon, 38700, La Tronche, France
sergiu.ivanov@univ-grenoble-alpes.fr

[6] School of Electric and Information Engineering,
Zhengzhou University of Light Industry,
Zhengzhou 450002, China

Summary. In the majority of models of P systems, rules are applied at the ticks of a global clock and their products are introduced into the system for the following step. In timed P systems, different integer durations are statically assigned to rules; time-free P systems are P systems yielding the same languages independently of these durations. In clock-free P systems, durations are real and are assigned to individual rule applications;

⋆ The work is supported by National Natural Science Foundation of China (61320106005 and 61033003) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012). This paper was finished with Rudolf Freund during Artiom Alhazov's and Sergiu Ivanov's stay in Vienna in August 2017.

⋆⋆ The work is supported by National Natural Science Foundation of China (61320106005 and 61033003) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

thus, different applications of the same rule may last for a different amount of time. In this paper, we formalise timed, time-free, and clock-free P system within a framework for generalised parallel rewriting. We then explore the relationship between these variants of semantics. We show that clock-free P systems cannot efficiently solve intractable problems. Moreover, we consider un-timed systems where we collect the results using arbitrary timing functions as well as un-clocked P systems where we take the union over all possible per-instance rule durations. Finally, we also introduce and study mode-free P systems, whose results do not depend on the choice of a mode within a fixed family of modes, and compare mode-freeness with clock-freeness.

1 Introduction

Membrane systems with symbol-objects are formal computational models of distributed multiset rewriting. While standard models often assume maximal parallelism and a global-clock synchronization of rules (overview in [12]), there have been a number of attempts in the literature to relax this condition. The extreme variant are so-called asynchronous systems, where the parallelism is arbitrary instead of maximal [1, 7]. Not surprisingly, in many cases such systems are much weaker (e.g., defining $PsMAT$ instead of $PsRE$) or need much stronger ingredients to be able to perform the same goal.

A different way to relax the global synchronisation condition is lifting the assumption that all rule executions take one step. For example, in timed P systems [5], a numerical function is defined, associating to each rule the positive integer number of steps its application takes. In this context, time-freeness is an (undecidable) property that the result of all computations of a P system does not depend on the timing function.

The motivation for studying time-freeness is investigating the power and the efficiency of P systems that are robust with respect to rule execution times. Yet, the definition of the time-freeness property is not restrictive enough for some goals—the time a rule application lasts cannot be different in different situations. Indeed, since the timing function is defined on the set of rules, the following facts are immediate:

1. If a rule is simultaneously applied multiple times, then all instances finish simultaneously.
2. If a rule is simultaneously applied in different membranes with the same label, then all rules finish simultaneously.
3. If a rule is applied at different steps of a computation, then all instances last for the same amount of time.
4. If a rule is applied in different non-deterministic branches of a computation, then all instances last for the same amount of time.

A number of publications investigate the efficiency of time-free P systems in solving intractable problems, e.g. [15, 16, 17, 18]. We believe that the constructions in these publications rely on the residual synchronisation facts listed above.

In this paper we focus on a variant of timing which allows individual rule executions to last differently. This variant was introduced under the name "clock-freeness" in [14]. In clock-free P systems, rule applications may last for "arbitrary" real periods and even applications of the same rule may have different durations. We prove that clock-freeness deprives any variant of P systems operating under this semantics of the capability of solving intractable (NP-complete) problems in polynomial time.

Clock-freeness changes the way in which a P system operates quite a bit. Indeed, since durations of rule applications are real numbers, such a P system does not follow the ticks of a global clock any more, but instead "listens" to events—situations in which rule applications finish and release new potential reactants (compare this to the preliminary observations in [11]). Such a P system therefore becomes event-driven and operates in continuous time, similarly to the Gillespie algorithm [10] or to data stream-driven reactive programs, e.g. [6]. In the present work, we formally define event-driven P systems and show their relationship to clock-free and time-free P systems. Moreover, we also introduce un-timed and un-clocked P systems, where as a result we take the union of all results obtained by any timing and per-instance timing function, respectively.

We also consider yet another freeness property: mode-freeness. A P system which is mode-free with respect to a family of modes has the same behaviour under all modes from this family. We show a large family of modes under which generating P systems yield trivial languages, but accepting P systems are computationally complete. Finally, we explore the form of some clock-free and mode-free P systems and show some relatively strong connections between the two freeness properties.

This article is organised as follows. Section 2 recalls some basic notions of formal language theory and then introduces a general definition of P systems rewriting objects from a computable set O. Section 3 recalls and formally defines timing functions, time- and clock-free P systems, as well as introduces event-driven P systems. Subsection 3.4 investigates the connections between these objects. Section 4 shows one of the main results of this paper: clock-free P systems cannot solve intractable problems in polynomial time. Section 5 introduces un-timed and un-clocked P systems. Section 6 recalls the notion of an evolution mode, introduces mode-freeness with respect to a family of modes, and then shows some properties of mode-free P systems. Section 7 compares clock-freeness with mode-freeness and points out some connections between these two properties. Section 8 discusses further possibilities for defining per-instance timing and clock-freeness. Section 9 concludes the paper and also lists several open problems.

2 Preliminaries

We assume the reader to be familiar with the basics of formal language theory and P systems, but we recall some of the notions for convenience. For further

introduction to the theory of formal languages and P systems, we refer the reader to [12, 13].

After recalling these basic notions, we will give a formal explanation of general rewriting in order to be able to introduce a general definition of P systems as hierarchical rewriting systems, somewhat in the spirit of [2] and [8].

2.1 Multisets

A *multiset* over V is any function $w : V \to \mathbb{N}$; $w(a)$ is the *multiplicity* of a in w. A multiset w is often represented by one of the strings containing exactly $w(a)$ copies of each symbol $a \in V$. The set of all multisets over the alphabet V is denoted by V°. By abusing string notation, the empty multiset is denoted by λ. We will also (ab)use the symbol $\in$ to denote the relation "is a member of" for multisets. Therefore, for a multiset w, $a \in w$ will stand for $w(a) > 0$.

Given two multisets $w, v \in V^\circ$, w is a submultiset of v if $w(a) \leq v(a)$, for all $a \in V$. In this case, removing w from v means constructing the multiset $v - w$ with the property $(v - w)(a) = v(a) - w(a)$.

For a multiset of tuples $w \in (A_1 \times \ldots \times A_n)^\circ$ we will use the notation $w|_{A_i}$ to refer to the multiset of projections of the elements of w on the dimension A_i, $1 \leq i \leq n$. Formally, $w|_{A_i} \in A_i^\circ$ and $w(a_i)$ for a fixed $a_i \in A_i$ is equal to the number of tuples of the form $(a_1, \ldots, a_i, \ldots, a_n)$ in w.

2.2 General Sequential and Parallel Rewriting

Consider an (infinite, computable) alphabet of objects O. An *O-rewriting rule* is a partial function $r : O \to O$. For an object $o \in O$ for which $r(o)$ is undefined, we say that r is not applicable to o. Often, the semantics of computing $r(o)$ is given by "removing the left-hand side" or r from the object o and then "adding back the right-hand side". Accordingly, we define the pair of partial functions $r_-, r_+ : O \to O$ such that their total effect is the same as that of r, i.e., $r = r_+ \circ r_-$.

Example 1. Consider the alphabet $V = \{a, b\}$ and the set $O = V^\circ$ of all finite multisets over V. The partial function $r : V^\circ \to V^\circ$ replacing an instance of a with two instances of b is an O-rewriting rule and is often written as $r : a \to bb$ or $r : a \to b^2$ (note that, in this notation, the symbol $\to$ is used to specify rule sides and not the domain or the codomain of a function); r is defined for all multisets containing at least an instance of a and is undefined for all other multisets.

For the multiset rewriting rule r, the value $r_-(w)$ can be defined by removing the left-hand side a from the multiset w (if possible) and $r_+(w)$ by adding the right-hand side bb to w. Thus, $r = r_+ \circ r_-$.

Example 2. Consider, again, the alphabet $V = \{a, b\}$ and the set $O = V^*$ of all finite strings over V. In this case, an O-rewriting rule is a partial function $r : O \to O$ replacing a substring *at a particular position*. To express the effect of rewriting *any* substring of a string $s \in O$ satisfying some particular criteria, we

need to consider a family of functions $(r_i : O \to O)_{i \in I}$ replacing the substring at its i-th occurrence. To express the effect of rewriting any substring in any finite string in O, we need to consider the family of functions $(r_i : O \to O)_{i \in \mathbb{N}}$.

Fix a set of O-rewriting rules R. To capture the possibility of applying multiple rules R in parallel, we define the (computable) partial function $apply : R^\circ \times O \to O$ which applies a multiset of rules from R to an object from O and yields a new object in O, if possible. As for the case of individual rules, to represent the idea of "removing the left-hand sides" and "adding the right hand sides", we define two other mappings $apply_-$ and $apply_+$ such that $apply(\rho, o) = apply_+(\rho, apply_-(\rho, o))$, with $\rho \in R^\circ$ and $o \in O$.

Example 3. Consider the alphabet $V = \{a, b\}$ and $O = V^\circ$, as in Example 1, and two rewriting rules $r_1 : ab \to bb$ as well as $r_2 : bb \to a$. Take the multiset of rules $\rho = r_1 r_2$; classically, the function $apply(\rho, w)$ is defined for such multisets $w \in O$ which contain the submultiset $ab^3 = ab\,bb$, necessary to satisfy both the applicability requirements of rules r_1 and r_2. In this case, $apply_-(\rho, w)$ is the function removing the multiset ab^3 from w, $apply_+(\rho, w)$ is the function adding $bb\,a$ to w, and $apply(\rho, w)$ is the function first removing ab^3 from w and then adding $bb\,a$.

A *sequential O-rewriting framework* is the pair (O, R), where O is a set of objects and R is a set of R-rewriting rules. A *parallel O-rewriting framework* is the pair $(O, R, apply_-, apply_+)$, where (O, R) is a sequential O-rewriting framework and $apply_-, apply_+ : R^\circ \times O \to O$ are the (computable) partial functions defining the semantics of parallel application of rules from R to objects in O.

Our definition of rewriting frameworks are strongly inspired by the work [8].

2.3 P Systems

The definition of P systems we give in this paper directly generalises various models of cell-like (hierarchical) P systems in which rules are "located within" the membranes and whose membrane structure may evolve: transition P systems with membrane dissolution rules, P systems with active membranes, etc.

A comprehensive overview of different flavors of membrane systems and their expressive power is given in the handbook which appeared in 2010, see [12]. For a state of the art snapshot of the domain, we refer the reader to the P systems website [20], as well as to the bulletin of the International Membrane Computing Society [19].

Given a parallel O-rewriting framework $(O, R, apply_-, apply_+)$, a P system is the following tuple:

$$\Pi = (O, O_T, \mu, w_1, \ldots, w_n, I, R_1, \ldots, R_n, h_i, h_o),$$

where O is a (computable, infinite) set of objects, $O_T \subseteq O$ is a (computable) set of terminal objects, μ is the initial membrane structure injectively labelled by

the numbers from $\{1,\ldots,n\}$ and usually given by a sequence of correctly nested brackets, I is the set of allowed *ingredients* (explained below), $w_i \in O$ is the initial object in membrane i, $R_i \subseteq R \times I$ is the set of O-rewriting rules associated with membrane i and enriched with some ingredients, $1 \leq h_i \leq n$ is the label of the input membrane and $1 \leq h_o \leq n$ is the label of the output membrane.

The set of ingredients I in the above definition captures the variety of additional actions which may be associated with O-rewriting rules. We give some examples:

- *Target indications*: If $O = V^\circ$, target indications can be represented by defining $I = \{none\} \cup (V \times Tar)^\circ$, thus allowing rules to specify multisets of pairs (a, tar) of symbols $a \in V$ and target indications $tar \in Tar$.
- *Membrane dissolution*: Membrane dissolution can be represented by defining $I = \{none, \delta\}$ and by writing non-dissolving rules as $(u \to v, none)$ and dissolving rules as $(u \to v, \delta)$, with the usual dissolution semantics.
- *Membrane division, creation, etc.*: Similarly to dissolution, any modification of the membrane structure may expressed by adding the corresponding symbols to the set I.

Finally, note that membrane polarisations can be represented without ingredients by extending the set of objects to $O \times \pi$, where π is the set of polarisations (e.g., $\pi = \{-, 0, +\}$), and by having the rules read and modify the polarisations if necessary.

A *configuration* of the P system Π is the tuple $C = (\mu', w'_1, \ldots, w'_n)$, where μ' is the current membrane structure and $w'_i \in O$ is the object contained in membrane i. For P systems which do not dynamically modify their membrane structure, the first component (μ') of the tuple may be omitted.

A *k-step computation of Π* is a sequence of configurations $(C_j)_{0 \leq j \leq k}$ with the following properties:

- $C_0 = (\mu, w_1, \ldots, w'_{h_i}, \ldots, w_n)$, where μ is the initial membrane structure of Π, w_i, $1 \leq i \leq n$, is the initial object in membrane i, and $w'_{h_i} = w_{h_i} \uplus w_{in}$, where $w_{h_i} \in O$ is the initial object in the input membrane h_i, $w_{in} \in O$ is the input object, and $\uplus$ is the operation of combining two objects (e.g., multiset union if $O = V^\circ$);
- for any configuration C_j, $0 \leq j < k$, the configuration C_{j+1} can be obtained from C_j by applying the rules to the objects of C_j according to a fixed *evolution mode* (e.g., the maximally parallel mode), and by then executing the actions required by the ingredients associated with the applied rules;
- C_k is a *halting configuration*, i.e., a configuration satisfying the halting condition of Π. One of the best known halting condition is requiring that no rule be applicable any more according to the fixed derivation mode (total halting by inapplicability).

The *result* of the computation $(C_j)_{1 \leq j \leq k}$ is derived from the object w_{h_o} found in membrane h_o in the halting configuration C_h. A typical way of deriving the result is applying the terminal projection $p_T : O \to O_T$ which allows for retrieving

the "terminal part" $p_T(w_{h_0})$. Another way may be declaring that the derivation $(C_j)_{1\leq j\leq k}$ only produces a result if $w_{h_o} \in O_T$ (otherwise Π produces no result).

P systems as we defined them are general device computing functions, yet particular cases are often considered. Π is said to work in the *generating mode* if it takes no input (the starting configuration C_0 is the same for all computations). Π is said to work in the *accepting mode* if it takes an input and accepts by a halting computation, whereas non-accepted inputs only yield non-halting computations.

A special case of accepting P systems are *deciding* P systems: for any input, all its computations must halt and are grouped into two classes—accepting and rejecting; for each input, all computations must belong to one of these groups. One usual way of discriminating between the two types of computation is by looking at the form of the object w_{h_o} in the output membrane in the halting configuration: e.g., if $O = V^\circ$, an accepting halting configuration of Π must contain the symbol *yes* in w_{h_o} and a rejecting halting configuration must contain the symbol *no*.

We will denote the language of objects generated (respectively, accepted) by the P system Π by $L_{gen}(\Pi)$ (respectively, $L_{acc}(\Pi)$). Sometimes we will use the notation $L(\Pi)$ when the context makes it clear whether Π is an acceptor or a generator.

3 Time- and Clock-freeness

In this section, we briefly recall (and generalise) the definition of timed and time-free P systems originally introduced in [5]. We then recall the original definition of clock-free P systems as introduced in [14] and give a formalisation. Finally, we show how clock-freeness can easily be captured via a simpler *event-driven semantics* (a natural continuation of [11]).

We start by defining the notion of a rule queue. Given a set of rules R, and the set of ingredients I, we will call any finite multiset of rules $\rho \in (R \times I)^\circ$ a *rule queue*. For a number set X, we will call any finite multiset $\rho \in (R \times I \times X)^\circ$ an *X-timed rule queue*. Intuitively, a rule queue is just an unordered collection of rules and ingredients, while an X-timed rule queue is a collection of rules and ingredients which have *timestamps*.

Given a P system Π, an *extended configuration* (with rule queues) is a tuple $C = (\mu, w_1, \ldots, w_n, \rho_1, \ldots, \rho_n)$, where $C = (\mu, w_1, \ldots, w_n)$ is a configuration of Π and ρ_i is a rule queue (with or without timestamps).

3.1 Time-free P Systems

We will now recall the definitions of timed and time-free P systems from [5] and generalise them to our definition of P systems.

Given a P system Π as defined in Subsection 2.3, a *timing function* is a computable mapping $e : R_\Pi \to \mathbb{N}_+$, with $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ and $R_\Pi = \bigcup_{1\leq i\leq n} R_i$, which assigns *durations* to the rules of Π. The *timed P system* $\Pi(e)$ is a P system with semantics modified in the following way.

Computations of Π are sequences of extended configurations with $\mathbb{N}_+$-timed rule queues (i.e., the rules in rule queues have natural timestamps). To compute the configuration C_{j+1} from a configuration C_j, consider the membrane i containing the object w_i and the rule queue ρ_i. $\Pi(e)$ shall perform the following actions:

1. Constitute the submultiset of rules ρ_{now} of the queue ρ_i which have the timestamp $j+1$; in other words, any tuple in ρ_{now} must have the form $(r, i, j+1)$. Build the new multiset ρ_i' by removing ρ_{now} from ρ_i. Take the multiset $\rho_{now}|_R$ of all O-rewriting rules in ρ_{now} and compute the object w_i' in the following way: $w_i' = apply_+(\rho_{now}|_R, w_i)$. Finally, implement the effects of all the ingredients listed in ρ_{now}.
2. Pick a multiset of rules ρ_{app} applicable to w_i' according to a fixed evolution mode and set the timestamp for every rule r added to ρ_{app} to $j + 1 + e(r)$. Take the multiset $\rho_{app}|_R$ of all O-rewriting rules in ρ_{app} and compute the new object w_i'' in the following way: $w_i'' = apply_-(\rho_{app}|R, w_i')$. Add ρ_{app} to ρ_i' thus constituting the new rule queue ρ_i''.
3. In configuration C_{j+1}, set the contents of membrane i to w_i'' and its rule queue to ρ_i''.

Thus, the queues in an extended configuration C_j contain the rules whose application started in the previous steps (excluding step j), including the rules which are scheduled to finish at step j. All queues are empty in the starting configuration and the first evolution step consists in launching some rules (in a sense, it is a "dummy" step or a "half step").

To halt, $\Pi(e)$ needs to exhaust all of the rule queues: that is, the evolution continues until there are still rules scheduled to finish in some future steps, and all queues must be empty in the halting configuration.

The result of a computation of the timed P system $\Pi(e)$ is derived from the contents of its output membrane in its halting configuration in the same way as described for non-timed P systems in Subsection 2.3.

A P system Π is called *time-free* if there exists a language of objects $L \subseteq O$ such that $L(\Pi(e')) = L(\Pi(e))$, *for any (computable)* timing functions $e : R_\Pi \to \mathbb{N}_+$ and $e' : R_\Pi \to \mathbb{N}_+$, and $L = L(\Pi(e))$ for some timing function $e : R_\Pi \to \mathbb{N}_+$. Therefore, time-freeness is the property of P systems to yield the same results independently of durations statically assigned to the rules.

3.2 Clock-free P Systems

In this subsection, we will formally define clock-free P systems following the original work [14]. The motivating intuition is as follows: real-world processes are rarely synchronised via a shared global clock. Timed and time-free P systems capture the fact that processes may have different durations and that some systems are robust to arbitrary variations in such durations; however the durations are integer numbers, which still implies the presence of a discrete global clock. Furthermore, in timed P systems, all applications of the same rule last for the same amount of

time, which does not take into account the variations in the execution time of different instances of the same process. Clock-free P systems as introduced in [14] lift both of these restrictions: different applications of *the same rule* are allowed to last for different, *real*, amounts of time.

Following the same scheme as for timed and time-free P systems, we can introduce per-instance real rule timing in the following way. Consider a P system Π as defined in Subsection 2.3 with O-rewriting rules enriched with ingredients $R_\Pi \times I$ and the set $\mathcal{C}$ of all sequences of extended configurations of Π. A *per-instance (real) timing function* is a mapping $\tau : \mathcal{C} \times (R_\Pi \times I)^\circ \to (R_\Pi \times I \times \mathbb{R}_+)^\circ$, with $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x > 0\}$, assigning positive real durations to the rules in a multiset of rules based on the given history of configurations.

Even before we define the effect of per-instance timing function, we give an informal example to give an intuitive impression.

Example 4. Consider the following one-membrane multiset rewriting P system:

$$\Pi_1 = (\{a, b\}^\circ, \{a, b\}^\circ, [\ ]_1, a, \{none\}, \{r_1 : a \to bb, r_2 : b \to aa\}, 1, 1)$$

and suppose it works in the maximally parallel mode. Take the initial configuration $C_0 = ([\ ]_1, aa, \lambda)$. Suppose we want to apply the rule $r_1 : a \to bb$ twice in this configuration. We will define the per-instance timing function τ to have the value $(r_1, none, 0.5)(r_1, none, \sqrt{2})$ for the singleton sequence (C_0) and the multiset of rules $(r_1, none)^2$. This will move the system into the configuration $C_1 = ([\ ]_1, \lambda, (r_1, none, 0.5)\,(r_1, none, \sqrt{2}))$.

Among the two applications of r_1, one is scheduled to finish earlier, at time 0.5. At this moment, it releases the multiset bb into the skin, which renders the rule r_2 applicable. We define the per-instance timing function τ to have the value $(r_2, none, \sin 1)(r_2, none, \cos 1)$ for the sequence (C_0C_1) and the multiset of rules $(r_2, none)^2$. This moves the system into the configuration $C_2 = ([\ ]_1, \lambda, \rho_2)$ with $\rho_2 = (r_1, none, \sqrt{2})(r_2, none, 0.5 + \sin 1)(r_2, none, 0.5 + \cos 1)$.

We will now define the semantics of per-instance real timing functions. Take a P system Π and fix a per-instance real timing function τ for it. Computations of $\Pi(\tau)$ are sequences of extended configurations with $\mathbb{R}_+$-timed rule queues (compare this with $\mathbb{N}_+$-timed rule queues for timed P systems recalled in Subsection 3.1). The queues in an extended configuration C_j contain the rules whose applications started in configurations previous to C_j (according to a fixed derivation mode), including rules scheduled to finish in this configuration. Consider a sequence $\gamma = (C_m)_{0 \le m \le j}$ of extended configurations with $\mathbb{R}_+$-timed rule queues. To compute the next configuration C_{j+1} from this sequence, $\Pi(\tau)$ proceeds in the following way:

1. Find the smallest timestamp $t_j \in \mathbb{R}$ across all rule queues in configuration C_j.
2. In every membrane i, take the submultiset ρ_{now} of the queue ρ_i in which the rules have the timestamp t_j and compute the object w_i' in the following way: $w_i' = apply_+(\rho_{now}|_R, w_i)$; also implement the effects of the ingredients listed in

ρ_{now}. Build ρ'_i by removing ρ_{now} from ρ_i. (This procedure is identical to that described in the semantics of timed P systems in Subsection 3.1, point 1.)

3. In every membrane i, pick a multiset of rules ρ_{app} applicable to w'_i according to a fixed evolution mode, compute $\rho'_{app} = \tau(\gamma, \rho_{app})$, add t_j to all timestamps in ρ'_{app}, and add the result to the new rule queue ρ'_i. Take the multiset $\rho_{app}|_R$ of all O-rewriting rules in ρ_{app} and compute the new object w''_i in the following way: $w''_i = apply_-(\rho_{app}|R, w'_i)$. Add ρ_{app} to ρ'_i, thereby forming the new queue ρ''_i. (This procedure is very similar to that described in the semantics of timed P systems in Subsection 3.1, point 2.)
4. In configuration C_{j+1}, set the contents of membrane i to w''_i and its rule queue to ρ''_i.

Like for timed P systems, the starting configuration of any computation of $\Pi(\tau)$ has all rule queues empty, and, to halt, $\Pi(\tau)$ needs to exhaust all queues.

The result of a computation of the P system $\Pi(\tau)$ equipped with the per-instance real timing function τ is derived from the contents of the output membrane in the halting configuration in the same way as described for non-timed P systems in Subsection 2.3.

A P system Π is called *clock-free* if there exists a language of objects $L \subseteq O$ such that $L(\Pi(\tau')) = L(\Pi(\tau))$, *for any (computable)* per-instance real timing functions τ and τ ', and $L = L(\Pi(\tau))$ for some per-instance real timing function τ. Therefore, clock-freeness is the property of P systems to yield the same results independently of positive real durations dynamically assigned to rule applications.

We will explicitly explain why our definition corresponds exactly to the slightly informal presentation given in [14]. In the cited paper, the author states that every rule application may have a different real duration. His proofs suppose durations may be arbitrary, but show computational completeness nevertheless. The fact that rule applications may have different real durations is captured by our per-instance real timing functions. Robustness with respect to arbitrary durations is captured by our definition of the clock-freeness property.

Example 5. Consider again the P system Π from Example 4 and the sequence of extended configurations (C_0, C_1, C_2). The corresponding evolution of the rule queue associated with the only membrane of Π is illustrated in Figure 1.

The hollow bullets on the time axis (denoted by the letter t on the figure) mark the "steps", i.e., the moments at which there is at least a rule which finishes its execution and when Π has to check whether any new rules have to be started. Clearly, the illustration does not show a halting computation of Π: new rules are started at moments $t = 0$ and $t = 0.5$, but, of course, other rules are applicable at the other moments highlighted in the figure. We do not show or treat them to avoid clutter.

Finally, we define an important class of per-instance real timing functions. We will call such a function τ *a Markovian real timing function* if its value does not depend on the first argument. Formally, τ is Markovian if, for a fixed multiset of

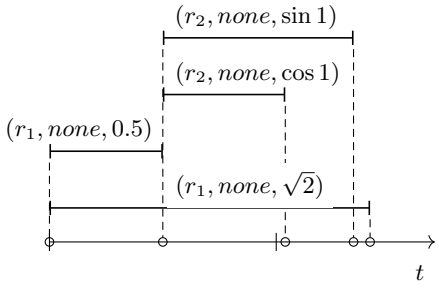


Fig. 1. A graphical illustration of the two-step computation of Π_1 described in Example 4. The hollow bullets mark the "steps".

rules ρ and for any two sequences of configurations γ_1 and γ_2, the following holds $\tau(\gamma_1, \rho) = \tau(\gamma_2, \rho)$. We will call P systems which are clock-free with respect to the class of Markovian timing functions *Markovian clock-free*.

3.3 Event-driven P Systems

Consider again the semantics of P systems with per-instance real timing functions, and especially the illustration in Figure 1. The evolution of such P systems is quite clearly centred around the concept of an *event*: the moment at which some rule executions finish and release the results into the membrane. The computations driven by per-instance real timing functions are essentially sequences of such events. This intuitively implies that what only matters is the order in which rules finish, and not so much the actual individual timings. This observation was stated in a preliminary form in [11].

In this section we first introduce event-driven P systems and then show the equivalence between this variant and clock-free P systems.

Following the same scheme as for per-instance real timing functions, we can define finishing functions in the following way. Consider a P system as defined in Subsection 2.3 with O-rewriting rules enriched with ingredients $R_\Pi \times I$ and the set $\mathcal{C}$ of all sequences of extended configurations of Π with simple rule queues (no timestamps). A *finishing function* is a mapping $\phi : \mathcal{C} \times (R_\Pi \times I)^\circ \to (R_\Pi \times I)^\circ$ indicating, based on the history of configurations, which rules from a given rule queue must finish their execution. Note that ϕ may also return an empty multiset.

Take a P system Π and fix a finishing function ϕ for it. We define the semantics of $\Pi(\phi)$ in the following way. Computations of $\Pi(\phi)$ are sequences of configurations with simple rule queues (no timestamps). Again, the rule queues of an extended configuration C_j contain the rules whose applications started before C_j according to the corresponding fixed derivation mode and $\Pi(\phi)$. Given a sequence $\gamma = (C_m)_{0 \leq m \leq j}$ of extended configurations with simple queues, $\Pi(\phi)$ proceeds in the following way to obtain the configuration C_{j+1}. In membrane i containing the object w_i and the rule queue ρ_i, Π does the following:

1. Apply the finishing function to find the submultiset of rules ρ_{now} which must finish: $\rho_{now} = \phi(\gamma, \rho_i)$. Take the multiset $\rho_{now}|_R$ of all rewriting rules in ρ_{now} and compute the new object $w_i' = apply_+(\rho_{now}|_R, w_i)$; also implement the effect of the ingredients listed in ρ_{now}. Build ρ_i' by removing ρ_{now} from ρ_i.
2. Pick a multiset of rules ρ_{app} applicable to w_i' according to a fixed evolution mode and add ρ_{app} to ρ_i', thereby constituting the new rule queue ρ_i''. Compute the new object w_i'' in the following way: $w_i'' = apply_-(\rho_{app}, w_i')$.
3. In configuration C_{j+1}, set the contents of membrane i to w_i'' and its rule queue to ρ_i''.

As before (Subsections 3.1 and 3.2), all computations start with empty rule queues and the system needs to exhaust all rule queues in order to halt. The result is retrieved as for P systems operating under conventional semantics (Subsection 2.3).

Recall that the finishing function ϕ is allowed to return an empty multiset. In this paper, we choose to only consider functions which, for a given sequence of configurations γ of a fixed P system Π, return a *non-empty multiset* for at least one rule queue (non-denying functions). This ensures that every configuration in a computation of Π corresponds to a rule finishing event.

Example 6. Consider again the P system from Example 4:

$$\Pi_1 = (\{a, b\}^\circ, \{a, b\}^\circ, [\ ]_1, a, \{none\}, \{r_1 : a \to bb, r_2 : b \to aa\}, 1, 1)$$

and the first three configurations of its computation (C_0, C_1, C_2) illustrated in Figure 1. We can reproduce the effects of these three steps using a finishing function. The initial configuration will be, as before, $K_0 = ([\ ]_1, aa, \lambda)$. In this configuration the maximally parallel mode forces Π to apply r_1 twice and to move into the following configuration $K_1 = ([\ ]_1, \lambda, (r_1, none)^2)$. We define the finishing function ϕ to return $(r_1, none)$ for history (K_0, K_1) and the queue $(r_1, none)$. This will release the products of r_1 into the skin membrane and render r_2 applicable. The maximally parallel derivation mode enforces the two applications of r_2, moving Π_1 into the configuration $K_2 = ([\ ]_1, \lambda, (r_1, none)\,(r_2, none)^2)$.

We now show side by side the configurations C_0, C_1, and C_2 of $\Pi(\tau)$ working under the per-instance real timing function τ from Example 4 and the configurations K_0, K_1, and K_2 from the previous example (we denote $t_1 = 0.5 + \sin 1$ and $t_2 = 0.5 + \cos 1$):

Note that, with the finishing function from Example 6, we are able to reproduce the contents of rule queues in (C_0, C_1, C_2), without using time stamps. We will later formally show that per-instance real timing functions are equivalent to finishing functions, which makes them into a useful instrument for reasoning about computations with per-instance real timing.

For a P system Π to be independent of the finishing strategy ϕ means that there exists a language $L \subseteq O$ of objects of Π such that $L = \Pi(\phi)$ for *any computable* finishing function ϕ.

	C_i	K_i
0	$([\ ]_1, aa, \lambda)$	$([\ ]_1, aa, \lambda)$
1	$([\ ]_1, \lambda,\ (r_1, none, 0.5)\,(r_1, none, \sqrt{2}))$	$([\ ]_1, \lambda,\ (r_1, none)^2)$
2	$([\ ]_1, \lambda,\ (r_1, none, \sqrt{2})\,(r_2, none, t_1)\,(r_2, none, t_2))$	$([\ ]_1, \lambda,\ (r_1, none)\,(r_2, none)^2)$

Table 1. A comparison between the forms of configurations in Examples 4 and 6. We denote $t_1 = 0.5 + \sin 1$ and $t_2 = 0.5 + \cos 1$.

Since a finishing function essentially defines the *sequencing* of the releases of "processed" rule products, P systems which are independent of this sequencing can be seen as "waiting" for events to happen and "handling" them. Thus, we will refer to such systems using the term *event-driven P systems*.

Finally, in analogy with Markovian per-instance timing functions, we define Markovian finishing strategies. We will call a strategy ϕ *a Markovian finishing strategy* if its value does not depend on the first argument. Formally, ϕ is Markovian if, for a fixed multiset of rules ρ and for any two sequences of configurations γ_1 and γ_2, the following holds: $\tau(\gamma_1, \rho) = \tau(\gamma_2, \rho)$. We will call P systems which are event-driven with respect the class of Markovian finishing functions *Markovian event-driven*.

3.4 Timing Types and Finishing Strategies

Because timed P systems, P systems with per-instance real timing, and P systems with finishing strategies stem from the same idea—introduce rule durations to P systems—it is not surprising that these models have a lot in common. In this subsection, we outline the main connections.

First of all, we would like to bring the reader's attention upon the form of the configurations shown in Table 1: in many of them, the multisets contained in the membranes are empty, the "semantic focus" being on rule queues. This is an effect which may be surprising at first, but which actually underlines the important difference of P systems with rule queues as compared to usual P systems: in the former case, configurations mark the *intervals* the start of some rule applications and the end of some other rule applications (also seen in Figure 1), while configurations for P systems operating under conventional semantics (Subsection 2.3) capture the *moments* between the end of some rule applications and the start of some other rule applications.

We will now show a series of intuitively clear inclusions of families of P systems with rule queues. We start with a general statement about timing functions and per-instance real timing functions.

Proposition 1. *Given a P system Π and any timing function e (Subsection 3.1) there exists a per-instance real timing function τ_e (Subsection 3.2) such that $L(\Pi(e)) = L(\Pi(\tau_e))$.*

Proof. Consider the timing function τ_e always assigning the duration $e(r)$ to any application of the rule r in any evolution of Π. It follows from the definitions of semantics in Subsections 3.1 and 3.2 that, for any computation γ_e of $\Pi(e)$ there exists a computation γ_τ of $\Pi(\tau_e)$ producing the same output object, and conversely. Moreover, for a given step j, the configurations $C_j \in \gamma_e$ and $K_j \in \gamma_\tau$ are identical (modulo the inclusion of $\mathbb{N}$ into $\mathbb{R}$).

The converse proposition is not true: there exist per-instance timing functions which do *not* have a corresponding timing function.

Proposition 2. *There exists a multiset-rewriting P system Π and a per-instance timing function τ such that $L(\Pi(\tau)) \neq L(\Pi(e))$ for any timing function e.*

Proof (Sketch). Consider the one-membrane multiset rewriting P system Π with the following rules:

$$r_1 : a \to c \qquad r_3 : cb \to d$$
$$r_2 : c \to f \qquad r_4 : cd \to \heartsuit$$

Fix the starting multiset in the only membrane of Π to aab. We can construct a per-instance real timing function for Π which will yield the evolution shown in Figure 2. Note that the two applications of r_1 take a *different* amount of time,

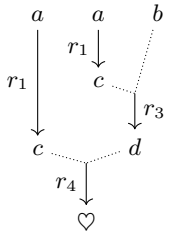


Fig. 2. A computation impossible without per-instance timing (because two applications of $r_1 : a \to c$ take different time).

which lets the first c arrive (on the right) to produce a d together with b so that, when the second c arrives (on the left), it can produce $\heartsuit$ together with d. On the other hand, if all applications of r_1 lasted for the same amount of time, both c's would appear at the same time, and one of them would have to evolve by rule r_3 turning into f and guaranteeing that rule r_4 cannot be applied.

The fact that in timed P systems in the sense of Subsection 3.1 different applications of the same rule last for the same amount of time implies the statement of the proposition.

According to the previous two propositions, per-instance timing allows richer behaviour than simple timing (in the sense of Subsection 3.1). This immediately implies the following statement.

Theorem 1. *A P system Π which is clock-free is also time-free.*

The relationship between per-instance timing functions and finishing strategies is even stronger. In the following, we say that two rule queues are *equivalent modulo timestamps* removing all timestamps from both yields two equal rule queues. We also consider the natural extension of this equivalence to configurations with rule queues.

Proposition 3. *Given an O-rewriting P system Π and any per-instance real timing function τ such that its value $\tau(\alpha, o)$ for the object $o \in O$ does not depend on the timestamps in the sequence of configurations α, there exists a finishing strategy ϕ such that $L(\Pi(\tau)) = L(\Pi(\phi))$.*

Proof. Take a computation γ of $\Pi(\tau)$ and suppose that we have already defined ϕ sufficiently to build a prefix $\bar{\gamma}'_j$ of length j of a computation γ' of $\Pi(\phi)$ in which all configurations are equivalent modulo timestamps to the corresponding configurations in the prefix $\bar{\gamma}_j$ of γ. Extend the definition of ϕ to require the same rules to finish in configuration K_j of γ' as those which are scheduled to finish in C_j in γ. Since we require τ to be independent of the timestamps in $\bar{\gamma}_j$, extending ϕ in this way is always possible. This observation, together with the fact that the starting configurations of γ and γ' are vacuously equivalent modulo timestamps (since their rule queues are empty), implies that we can define ϕ such that all configurations in γ' are equivalent modulo timestamps to the corresponding configurations in γ. This means that the results in the halting configurations of γ and γ' are equal, which proves the proposition.

Corollary 1. *Given a P system Π and any Markovian per-instance real timing function τ, there exists a finishing strategy ϕ such that $L(\Pi(\tau)) = L(\Pi(\phi))$.*

This corollary directly implies the following statement about event-driven P systems and Markovian clock-free P systems.

Theorem 2. *Any P system Π which is event-driven is Markovian clock-free.*

The converse of Proposition 3 also holds.

Proposition 4. *Given a P system Π and any finishing strategy ϕ, there exists a per-instance real timing function τ such that $L(\Pi(\phi)) = L(\Pi(\tau))$.*

Proof. Take a computation γ of $\Pi(\phi)$; γ is a sequence of configurations with simple rule queues (without timestamps). Construct a new sequence of configurations γ' with $\mathbb{R}_+$-timed rule queues in which all rules in all rule queues of configuration C_j get the timestamp j. Now consider the per-instance real timing function τ which assigns exactly these timestamps to rule applications in γ'. The fact that we can always carry out this transformation implies the statement of the proposition.

According to the previous proposition, per-instance timing strategies may ensure richer behaviour than finishing strategies, which implies the following statement.

Theorem 3. *A P system Π which is clock-free is event-driven (in the sense of Subsection 3.3).*

Figure 3 summarises the relations between the various kinds of freeness properties of P systems with rule queues we have considered in this paper. This figure

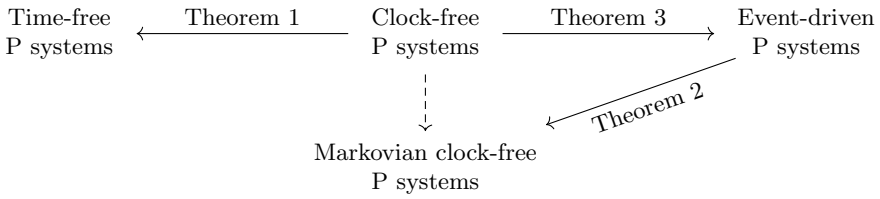


Fig. 3. Inclusions between the different kinds of freeness properties considered in this section.

also takes into consideration that any clock-free P system is trivially Markovian clock-free (because Markovian per-instance timing functions form a proper subclass of per-instance timing functions). This relation is represented as a dashed arrow.

4 Clock-freeness and Efficiency: P versus NP

One of the famous features of some variants of P systems is the capability of solving intractable (NP-complete) problems in polynomial time. The classical approach is generating an exponential number of computing units in polynomial time, which allows fast exploration of the space of candidate solutions (see [12] for some classic examples). Recently, efficient *time-free* solutions to intractable problems have been provided, e.g. in [15, 16, 17, 18]. Since there is no upper bound on the values the timing function may assign, the authors of the cited papers measure the time complexity of their constructions in terms of rule starting steps—the number of moments in the evolution of the P system at which rule executions start—rather than in terms of the total running time.

On the other hand, we tend to see the number of *rule finishing steps* as a better measure for time complexity of time- and clock-free P systems. We take as a motivating example a computation in which rules only start in the first configuration and then finish at different times. This computation has only one starting step, but

it may have multiple finishing steps, the number of which is more closely related to the number of events that occurred.

We now show that, assuming that in order to solve an NP-complete problem an exponential number of computing units is necessary, efficient (in terms of the number of finishing steps) *clock-free* solutions are impossible to construct (we assume $\mathsf{P} \neq \mathsf{NP}$). The intuition is as follows: in time-free P systems, we may not assume any particular duration for a given rule application, but we are sure that all applications of the same rule take the same amount of time. The fact that this property is no longer guaranteed under clock-freeness turns out to be essential for (in)efficiency.

Theorem 4. *Consider an* NP*-complete problem $\mathcal{P}$ and take a P system Π solving it. Then there exists a per-instance (real) timing function under which all computations of Π contain an exponential (in the size of the input) number of rule finishing steps (assuming that in order to solve an* NP*-complete problem an exponential number of computing units is needed).*

Proof. In P systems as defined in Subsection 2.3, the atomic "computing units" are single rule applications. Therefore, according to our assumption, Π must run an exponential number of rule applications. Since Π operates under per-instance timing, we can ensure that no two rule applications end at the same time, which implies that Π has exponentially many rule finishing steps.

5 Un-timed and Un-clocked P Systems

In contrast to time-free P systems, where all timing functions have to generate the same results, in an un-timed P systems we collect all the results obtained by using any timing function, i.e., for a given P system Π we define

$$L_{un-timed}(\Pi) = \bigcup_{t \text{ timing function}} L(\Pi, t).$$

Moreover, in the same way, in an un-timed P system we collect all the results obtained by using any per-instance timing function, i.e., for a given P system Π we define

$$L_{un-clocked}(\Pi) = \bigcup_{\tau \text{ per-instance timing function}} L(\Pi, \tau).$$

In an un-clocked P system, we simply may assume each rule application to last an arbitrary amount of time. In the following we give a small example which yields different results when considered as an un-clocked or as an un-timed system and is neither time- nor clock-free:

Example 7. We consider the one-membrane multiset rewriting P system Π with the following non-cooperative rules with inhibitors:

$$r_1 : a \to aa|_{\neg c} \quad \text{and} \quad \mathrm{r}_2 : \mathrm{b} \to \mathrm{c}$$

Starting from the axiom ab, in parallel we have to apply both $r_1 : a \to aa|_{\neg c}$ and $b \to c$ in parallel. Considering Π as an un-timed system, r_1 can be applied again and again in parallel to all symbols a being generated until the application of the rule r_2 has finished which immediately stops the derivation by the appearance of the inhibitor c. In sum, we obtain

$$L_{un-timed}(\Pi) = \bigcup_{t \text{ timing function}} L(\Pi, t) = \bigcup_{n \in \mathbb{N}} \{a^{2^n} c\}.$$

Of course, this system is neither time- nor clock-free. The infinite set is generated not due to the choice between applicable rule multisets, but due to the non-deterministic choice of the timing function.

Considering Π as an un-clocked system, again $r_1 : a \to aa|_{\neg c}$ and $b \to c$ have to be applied in parallel in the first step, and r_1 can be applied until the application of the rule r_2 has finished which immediately stops the derivation by the appearance of the inhibitor c. Yet in contrast to the un-timed version, the applications of the rule r_1 to the symbols a appearing in the meantime may end at arbitrary moments of time. To the two symbols a appearing when the first application of rule r_1 has finished, r_1 has to be applied simultaneously to both symbols a, yet from that moment on the different instances of rule r_1 may finish in an unsynchronized way. Hence, as we sum up all possible results, we may restrict ourselves to consider only the events when just one rule application ends. The only symbols to which a rule, i.e., r_1, now can be applied are the two symbols a having evolved as a result of this one rule application, and to these two symbols two copies of r_1 have to be applied simultaneously. In fact, this only means that the number of symbols a has increased by one with each finishing of a rule r_1. Therefore, in sum we obtain

$$L_{un-clocked}(\Pi) = \bigcup_{\tau \text{ per-instance timing function}} L(\Pi, \tau) = \{a^n c \mid n \in \mathbb{N}_+ \setminus \{1, 3\}\}.$$

A similar result can be obtained by the one-membrane multiset rewriting P system Π' with the following non-cooperative rules with promoters:

$$r_1 : a \to aa|_b, \quad r_2 : b \to b, \quad \text{and} \quad \mathrm{r}_3 : \mathrm{b} \to \mathrm{c}$$

Starting from the axiom ab, we now may assume that the execution of the rules r_1 and r_2 takes exactly the same time, because the promoter b is needed to allow the copies of rules r_1 to be applied. Again, the derivation halts as soon as the promoter b is eliminated by applying r_3. For the un-timed mode, the application of rules r_1 still is synchronized, too, and we therefore obtain

$$L_{un-timed}(\Pi') = \bigcup_{t \text{ timing function}} L(\Pi', t) = \bigcup_{n \in \mathbb{N}} \{a^{2^n} c\}.$$

In the un-clocked mode, the finishing of rules may be arbitrary, yet still the promoter b is needed to continue with applying rule r_1, i.e., only when the application of the rule $r_2 : b \to b$ has finished, rule r_1 can be applied. In sum, we again obtain

$$L_{un-clocked}(\Pi') = \bigcup_{\tau \text{ per}-\text{instance timing function}} L(\Pi', \tau) = \{a^n c \mid n \in \mathbb{N}_+ \setminus \{1, 3\}\}.$$

6 Mode-freeness

Considering time-free, clock-free, and event-driven P systems motivates further discussion about robustness with respect to variations of other parameters. In this section we will consider *mode-freeness*: robustness with respect to the choice of the evolution mode.

6.1 Evolution Modes

Take a (computable) set of objects O and consider a parallel O-rewriting framework $(O, R, apply_-, apply_+)$. Following [9], we denote by $Appl(R, o)$ the set of multisets of rules applicable to the object $o \in O$ in parallel. Given an n-membrane O-rewriting P system Π and a configuration C of it, we denote by $Appl(\Pi, C)$ the set of tuples of the form $(\rho_1, \ldots, \rho_n)$, in which ρ_i is a multiset of rules applicable to the object w_i in membrane i in configuration C.

Example 8. Consider the multiset ab and the set of multiset rewriting rules $R = \{r_1 : a \to b, r_2 : b \to c\}$. Then $Appl(R, ab) = \{r_1, r_2, r_1 r_2\}$. Take a multiset-rewriting P system Π with the set of ingredients $I = \{none\}$ and two membranes with equal sets of rules $R_1 = R_2 = \{(r_1, none), (r_2, none)\}$. Consider the configuration $C = (\mu, ab, ab)$ of Π, then $Appl(\Pi, C) = A \times A$, where $A = \{(r_1, none), (r_2, none), (r_1, none)(r_2, none)\}$.

Given a P system Π and a configuration C, an *evolution mode* (derivation mode) is a strategy ϑ for filtering the set $Appl(\Pi, C)$. According to [9], we denote by $Appl(\Pi, C, \vartheta) \subseteq Appl(\Pi, C)$ the set of tuples of multisets of rules of Π applicable in configuration C according to the derivation mode ϑ. When $Appl(\Pi, C, \vartheta)$ contains more than one element, Π chooses between the allowed tuples nondeterministically in order to continue the computation. We will denote the language generated (respectively, accepted) by Π operating under the derivation mode ϑ by $L_{gen}(\Pi, \vartheta)$ (respectively, $L_{acc}(\Pi, \vartheta)$).

We will now recall some typical examples of derivation modes considered in [9]. All of these examples are formulated for a P system Π and a configuration C of it.

Example 9. The *asynchronous* derivation mode *asyn* is the mode allowing any combination of rules to be applied: $Appl(\Pi, C, asyn) = Appl(\Pi, C)$.

Example 10. The *sequential* derivation mode *sequ* is the mode only allowing one rule to be applied at any time. $Appl(\Pi, C, sequ)$ therefore contains tuples of singleton multisets of rules.

Example 11. The *maximally parallel* derivation mode *max* only includes tuples of non-extendable multisets of rules.

Formally, for a tuple $(\rho_1, \ldots, \rho_n) \in Appl(\Pi, C, max)$, the set $Appl(\Pi, C)$ contains *no* tuple $(\rho'_1, \ldots, \rho'_n)$ such that at least one ρ_i is a submultiset of ρ'_i, for $1 \leq i \leq n$.

A very interesting derivation mode (considered in a detailed way in [3]) is the following one.

Example 12. The *set-maximally parallel* mode *smax* only allows tuples of multisets containing at most one instance of any rule. Formally, for any tuple $(\rho_1, \ldots, \rho_n) \in Appl(\Pi, C, smax)$, it is true that $\rho_i(r) \leq 1$ for any rule r in membrane i, $1 \leq i \leq n$.

Finally, we show several more derivation modes which we use later.

Example 13. The $max_{\geq k}$ mode only allows tuples of multisets which contain *at least k* rules. That is, for any tuple $(\rho_1, \ldots, \rho_n) \in Appl(\Pi, C, max_{\geq k})$, it must hold that $|\rho_i| \geq k$, for all $1 \leq i \leq k$.

Example 14. Suppose that the sets of rules R_i associated with membranes i of Π, $1 \leq i \leq n$, are equipped with total orders $\leq_i$ and consider the mode *det* ("the determinator") which only allows tuples of singleton multisets of rules, which are also minimal with respect to the corresponding order. Formally, for any tuple $(\rho_1, \ldots, \rho_n) \in Appl(\Pi, C, det)$, it is true that $|\rho_i| \leq 1$ and, for any other tuple of singleton multisets $(\rho'_1, \ldots, \rho'_n) \in Appl(\Pi, C)$, it holds that $r_i \leq_i r'_i$, where $\rho_i = r_i$, $\rho'_i = r'_i$, and $1 \leq i \leq n$.

Note that, according to this definition, $Appl(\Pi, C, det)$ is either empty (if $Appl(\Pi, C)$ is empty) or a singleton set, which justifies the informal name "the determinator".

Finally, an extreme example of an evolution mode.

Example 15. The *empty* evolution mode $\varnothing$ is the evolution mode disallowing any rule applications: $Appl(\Pi, C, \varnothing) = \emptyset$.

6.2 Freeness with Respect to a Family of Modes

Consider an O-rewriting P system Π and the family of evolution modes Θ. We say that Π is *Θ-mode-free* if there exists a language $L \subseteq O$ such that $L = L(\Pi, \vartheta)$, *for all* $\vartheta \in \Theta$. We use the notation $pL_{gen}(O, \Theta)$ to refer to the family of languages over O generated by Θ-mode-free O-rewriting P systems. We replace the subscript *gen* by *acc* to refer to the family of languages accepted by Θ-mode-free O-rewriting P systems.

It turns out that the idea mode-freeness has already been indirectly invoked in the literature. Indeed, the constructions from the paper [3] that literally hold for the modes max and $smax$ are $\{max, smax\}$-mode free.

We start our discussion of more general kinds of mode-freeness by remarking that mode-freeness with respect to the family of all modes (denoted by Θ_U) is a very restrictive condition filtering out non-trivial behaviour.

Proposition 5. *Consider the family of all derivation modes Θ_U. Then the following statements hold:*

- $pL_{gen}(O, \Theta_U)$ *only contains* $\emptyset$ *and singleton languages,*
- $pL_{acc}(O, \Theta_U) = 2^O$*, where* 2^O *is the set of all subsets of* O*, but all computation is done by the procedure extracting the result from the output object.*

Proof. Consider the Θ_U-mode free O-rewriting P system Π. Since Π should yield the same language under *any* mode, it is sufficient to investigate its behaviour under the empty mode $\varnothing$. Under this mode, Π never evolves.

For generation, this means that the result is computed from the initial object placed in the output membrane, which, depending on the procedure for extracting the result, may yield a singleton language or the empty language (e.g., in the case in which the initial object in the output membrane has no corresponding terminal projection).

Suppose now that Π is an acceptor. We will consider the following cases.

- If Π accepts by halting, it accepts any object because it halts immediately: $L_{acc}(\Pi) = O$.
- Suppose Π accepts by placing an object of a specific form into the output membrane.
  - If the output membrane of Π is different from its input membrane and the initial object placed into the output membrane does not satisfy the acceptance criterion, then Π rejects all inputs: $L_{acc}(\Pi) = \emptyset$.
  - If the output membrane of Π is the same as its input membrane, then Π will accept those inputs objects which satisfy the acceptance criterion for output objects. Therefore Π can be made to accept any subset of O by varying its acceptance criterion.

These observations conclude the proof.

To avoid trivial results, we assume in what follows that the procedure for extracting the result out of the output object is reasonably simple.

As we have just seen, Θ_U-mode-freeness is a very strong restriction. We will now consider an important subfamily of Θ_U: non-denying modes. Given a P system Π, a mode ϑ is *non-denying* if, for any configuration C of Π, $Appl(\Pi, C) \neq \emptyset$ implies that $Appl(\Pi, C, \vartheta) \neq \emptyset$. The mode is called denying otherwise. We will use the notation $\Theta_{\neg deny}$ to refer to the subfamily of non-denying modes.

Example 16. The derivation modes *asyn*, *sequ*, *max*, *smax*, and *det* are non-denying. The derivations modes $max_{\leq k}$ and $\varnothing$ are denying modes.

Mode-freeness with respect to non-denying modes turns out to be a much more interesting property than Θ_U-mode-freeness. In the generative case, $\Theta_{\neg deny}$-mode-freeness yields P systems generating singleton languages and the empty language. (Note that Θ_U-mode-free P systems can achieve the same behaviour only by playing on the procedure for extracting the result out of the output object.)

Proposition 6. *$pL_{gen}(O, \Theta_{\neg deny})$ only contains $\emptyset$ and singleton languages.*

Proof. Consider a $\Theta_{\neg deny}$-mode-free P system Π. Since Π should generate the same language under any non-denying mode, we can investigate its behaviour under "the determinator" mode *det*. Under this mode, Π evolves sequentially and deterministically. This means that it can either generate a singleton language, or the empty language in case it never halts or generates an output object without a terminal projection.

The situation changes drastically in the accepting case: indeed, deterministic acceptor P systems simulating deterministic register machines exist (e.g., [4, Theorem 2]). We nevertheless sketch a simple construction here.

Theorem 5. *Given an alphabet V and the set of recursively enumerable multiset languages $RE \subset V^\circ$, the following holds: $pL_{acc}(V^\circ, \Theta_{\neg deny}) = RE$.*

Proof (Sketch). Consider the one-membrane multiset-rewriting P system Π simulating a deterministic register machine M. Π uses cooperation and inhibitors. For every instruction $p : (A(r), q)$ incrementing register r and going from state p to state q, Π has a rule $p \to qr$. For every instruction $p : (S(r), q, z)$ checking register r for zero in state p, decrementing r and moving into state q, or moving into state z if the decrement is not possible, Π includes the rule $pr \to q$ to ensure the decrement and the rule $p \to z|_{\neg r}$ for the zero check.

Two properties follow from this construction sketch:

- Π correctly simulates the computations of M,
- exactly one rule is applicable in any evolution step of Π, which means that Π is sequential and deterministic.

The second property implies that, if we take $L = L_{acc}(\Pi, det)$, then $L = L_{acc}(\Pi, \vartheta)$ for any non-denying derivation mode $\vartheta \in \Theta_{\neg deny}$, i.e., Π is $\Theta_{\neg deny}$-mode-free. The fact that we can perform this construction for any register machine M implies the statement of the theorem.

The two previous statements highlight a huge gap between the generating and the accepting cases under mode-freeness with respect to non-denying modes: mode-free generation only produces trivial languages, while mode-free acceptance is computationally complete.

The construction in Theorem 5 allows us to derive a sufficient criterion for mode-freeness with respect to non-denying modes.

Theorem 6. *If a P system Π is deterministic under the evolution mode asyn in any reachable configuration, then it is $\Theta_{\neg deny}$-mode-free.*

Proof. For Π to be deterministic under *asyn* means that, for any reachable configuration C, the set $Appl(\Pi, C, asyn)$ is a singleton set. This only happens when at most one rule is applicable to C in the whole system. The effect of all non-denying modes on Π will therefore be the same: apply the only applicable rule (if there exists one), or halt if no more rules are applicable anywhere. This observation implies that Π is $\Theta_{\neg deny}$-mode-free.

The converse statement is not necessarily true in general: for example, a multiset-rewriting P system whose only behaviour consists in erasing all the symbols in the input one by one will be able to behave similarly under any non-denying mode. We do expect the converse statement to be true for "reasonable" P systems, however.

Conjecture 1. Any computationally universal $\Theta_{\neg deny}$-mode-free P system is deterministic under *asyn* in any of its reachable configurations.

We recall that being computationally universal means being capable to "run any program". More concretely, a P system is computationally universal if it can simulate a universal register machine. We refer the reader to [12] for comprehensive explanations.

7 Clock-freeness versus Mode-freeness

In this section we start a discussion about the relationship between clock- and mode-freeness. Despite their different origins, the two freeness properties exhibit a number of similarities. Consider, for example, the sketch of the $\Theta_{\neg deny}$-mode-free P system simulating an arbitrary register machine from Theorem 5. This system is trivially clock-free because at most one rule can be applied at any time. Furthermore, we can reformulate the criterion from Theorem 6 for the clock-free case in the following way.

Theorem 7. *If a P system Π is deterministic under the evolution mode asyn in any reachable configuration, then it is clock-free.*

Proof. By the same arguments as in the proof of Theorem 6, we conclude that a P system Π with the required properties may only apply at most one rule at any step, which trivially implies clock-freeness.

In case Conjecture 1 is true, being $\Theta_{\neg deny}$-mode-free is equivalent to being deterministic under the mode *asyn* for computationally universal P systems. This allows us to formulate the following derived hypothesis.

Conjecture 2 (assuming Conjecture 1). Any computationally universal $\Theta_{\neg deny}$-mode-free P system is also clock-free.

The two statements we have formulated in this section reveal parts of a strong relationship between clock- and mode-freeness. In particular, the previous conjecture warrants wondering whether any clock-free P system is also $\Theta_{\neg deny}$-mode-free. The following example and the associated propositions allow us to answer this question in the negative.

Example 17. Consider a register machine M and construct a one-membrane multiset rewriting P system Π_M in the following way.

- For every instruction $p : (A(r), q, q')$ which increments register r and non-deterministically moves from state p to either state q or q', add the rules $p \to qr$ and $p \to q'r$ to Π_M.
- For every instruction $p : (S(r), q, z)$ decrementing r and moving into state q, or moving into state z if the decrement is not possible, add the following rules to Π_M:

$$p \to p'p_r$$

$$\begin{array}{ll} p' \to p'' & p_r r \to d_r \\ p''p_r \to z & p''d_r \to q \end{array}$$

Π_M operates under the maximally parallel mode, under normal semantics (no timing, finishing strategies, etc.), and simulates the register machine M. Simulation of the increment instruction is straightforward. To simulate the decrement, Π_M splits the state symbol p into p' and p_r. p_r tries to decrement the register r by the rule $p_r r \to d_r$ while p' waits for one step turning into p''. Then, if p'' finds a d_r (meaning that the register was successfully decremented), the rule $p''d_r \to q$ is applied; otherwise the rule $p''p_r \to z$ is applied ensuring the correct choice between states q and z.

Proposition 7. *Π_M operating under the [set-]maximally parallel mode is clock-free (and event-driven).*

Proof. The only moment at which Π_M applies more than one rule is during the simulation of the decrement, when the symbols p' and p_r are produced and the configuration contains an instance of r. In this case, p' and p_r are immediately consumed by the applications of the corresponding rules (because Π_M operates in the [set-]maximally parallel mode). Note that no rule in Π_M is applicable before *both* p'' *and* d_r are produced, which makes the behaviour of Π_M independent of the timings on the individual applications of rules in this branch of the simulation. These observations imply that Π_M is clock-free.

Proposition 8. *Π_M is not $\Theta_{\neg deny}$-mode-free.*

Proof. As seen in the proof of the previous proposition, Π_M operating under the modes *max* and *smax* simulates the register machine M. However, if we fix a total order on the rules of Π_M such that $p' \to p''$ is less than $p_r r \to d_r$, and have Π_M operate under "the determinator" mode *det*, then Π_M will never have the chance to apply the rule $p_r r \to d_r$, meaning that Π_M will not simulate M any more.

This shows that the language accepted/generated by Π_M is different under two different non-denying modes (max and det) which implies the statement of the theorem.

Corollary 2. *There exists a clock-free P system which is not $\Theta_{\neg deny}$-mode-free.*

8 A Note on the Semantics of Clock-freeness

We would now like to use the instruments we have constituted throughout the paper to point out some issues with the informal introduction of clock-freeness in the original work [14]. These issues appear under derivation modes different from the maximally parallel one—*extendable modes*—or with rules which have *non-monotonous* rule applicability semantics. We now define these terms formally.

Given a P system Π and a configuration C of Π, we will call a mode ϑ *non-extendable* if all multisets in $Appl(\Pi, C, \vartheta)$ are non-extendable, i.e., for any $\rho \in Appl(\Pi, C, \vartheta)$, there exists *no* $\rho' \in Appl(\Pi, C)$ such that ρ is a submultiset of ρ'. If ϑ is not non-extendable, it is called extendable.

Example 18. The maximally parallel mode is by definition a non-extendable mode, but any other mode ϑ such that $Appl(\Pi, C, \vartheta) \subseteq Appl(\Pi, C, max)$ is non-extendable as well.

Given a parallel rewriting framework $\mathcal{F} = (O, R, apply_-, apply_+)$ and a partial order relation $\subseteq$ on O, we say that the rule applicability semantics of $\mathcal{F}$ is *monotonous* if, for two objects $o_1, o_2 \in O$, $o_1 \subseteq o_2$ implies that $Appl(R, o_1) \subseteq Appl(R, o_2)$, where $Appl(R_1, o)$ denotes the set of multisets of rules from R applicable to o_1.

Example 19. The semantics of cooperative multiset rewriting is monotonous: for a fixed set of multiset rewriting rules R and two multisets w_1 and w_2 such that w_1 is a submultiset of w_2, at least as many rules are applicable to w_2 as to w_1.

The semantics of cooperative multiset rewriting rules with *inhibitors* is non-monotonous: consider the singleton set of rules $R = \{r : a \to b|_c\}$ and the multisets $w_1 = a$ and $w_2 = ac$; w_1 is a submultiset of w_2, but r is only applicable to w_1 and not to w_2.

Now consider again the informal definition of clock-free semantics from [14] and take a P system Π with non-monotonous rule applicability semantics. Whenever some rules can be applied, Π has to start their application by "removing their left-hand sides" using $apply_-$. However, since the applicability semantics in Π is non-monotonous, this may immediately render more rules applicable. Letting Π continue "removing the left-hand sides" would mean that Π may run parts of the computation which should follow each other *at the same moment*.

Suppose now that Π works under a mode ϑ which is extendable. This means that Π does *not* have to start all of the rules which are potentially applicable

immediately. Since Π does not have a global clock, we do not know when Π should consider applying these left-over rules, and if it starts applying them immediately, it would violate the derivation mode ϑ.

The definitions in Subsection 3.2 address both of these issues by declaring that the only time Π should start new rule applications is when some (other) rule applications release new products. Another way of handling these problems would be restricting per-instance real timing functions to only take values in a closed interval $[c_0; +\infty) \subseteq \mathbb{R}_+$, for some fixed positive constant $c_0 \in R_+$. Under this restriction, we know that any rule takes at least c_0 units of time to finish, which means that Π could reconsider applying new rules either when some rule products become available, or c_0 units of time after the last pack of rule applications started.

Changing the way in which per-instance timing is defined should give rise to formulations of different event-driven semantics. Indeed, with the restriction described in the previous paragraph, the types of events to which Π may react would be extended with the ticks of a "local timer" going off in c_0 units of time after each start of some rule applications.

9 Conclusion and Discussion

In this paper we recalled timed, time-free, and clock-free P systems and provided a common framework for the three notions. This framework allows discussing different kinds of timing functions and freeness properties for P systems operating on arbitrary object types allowing for parallel rule application. We also discussed mode-freeness and showed that, even though mode-freeness and clock-freeness express robustness with respect to variations in quite different parameters, mode-free and clock-free P systems exhibit a number of similarities.

Both mode-freeness and clock-freeness as well as the other concepts like in un-timed and un-clocked P systems seem to offer plenty of possibilities for future research, among which we would like to state the following ones, in no particular order.

1. *Complete Figure 3:* Further investigate the relationship between the freeness properties shown in Figure 3. Find new inclusions, show the (non-)strictness of the known inclusions, consider yet different variations of the timing functions and finishing strategies.
2. *Prove or disprove Conjecture 1:* This conjecture states that that any $\Theta_{\neg deny}$-mode-free P system is *asyn*-deterministic in any reachable configuration C. As shown in Conjecture 2, this could reveal a strong connection between mode-free P systems and clock-free ones.
3. *Other families of modes:* We have only considered two infinite families of evolution modes in detail—the family of all modes Θ_U and the family of non-denying modes $\Theta_{\neg deny}$. We showed that the properties of P systems being mode-free with respect to these families are rather unusual (huge gap between the power

of generators and acceptors). Are there other families of modes exhibiting similar properties?

4. *Halting conditions:* In this paper we essentially glossed over halting conditions. Investigating mode-freeness and clock-freeness with respect to different halting conditions may prove interesting. What would freeness with respect to some families of halting conditions mean?
5. *Mode-freeness without inhibitors:* Theorem 5 shows a computationally complete family of $\Theta_{\neg deny}$-mode-free P systems. These P systems rely on cooperativity *and* on inhibitors (as usual, priorities would work just as well). What are the languages accepted by multiset-rewriting $\Theta_{\neg deny}$-mode-free P systems without inhibitors?
6. *Different clock-freeness:* As pointed out in Section 8, the intuitive idea of allowing individual rule applications to last for a different amount of time gives rise to multiple possible semantics. Subsection 3.2 describes one of them; exploring other possibilities may prove interesting for applications in modelling.
7. *Un-timed and un-clocked P systems:* Which variants of P systems still remain computationally complete when being considered as un-timed or un-clocked systems?

References

1. Artiom Alhazov and Rudolf Freund. Asynchronous and maximally parallel deterministic controlled non-cooperative P systems characterize NFIN and confin. In Erzsébet Csuhaj-Varjú, Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and György Vaszil, editors, *Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*, volume 7762 of *Lecture Notes in Computer Science*, pages 101–111. Springer, 2012.
2. Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, and Marion Oswald. Observations on P systems with states. In Marian Gheorghe, Ion Petre, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Multidisciplinary Creativity. Hommage to Gheorghe Păun on His 65th Birthday*. Spandugino, 2015.
3. Artiom Alhazov, Rudolf Freund, and Sergey Verlan. P systems working in maximal variants of the set derivation mode. In Alberto Leporati, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers*, volume 10105 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2016.
4. Cristian S. Calude and Gheorghe Păun. Bio-steps beyond Turing. *BioSystems*, 77(1-3):175–194, November 2004.
5. Matteo Cavaliere and Dragoş Sburlan. Time–independent p systems. In Giancarlo Mauri, Gheorghe Păun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing: 5th International Workshop, WMC 2004, Milan, Italy, June 14-16, 2004, Revised Selected and Invited Papers*, pages 239–258. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
6. Conal Elliott and Paul Hudak. Functional reactive animation. In *International Conference on Functional Programming*, 1997.

7. Rudolf Freund. Asynchronous P systems and P systems working in the sequential mode. In Giancarlo Mauri, Gheorghe Paun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC 2004, Milan, Italy, June 14-16, 2004, Revised Selected and Invited Papers*, volume 3365 of *Lecture Notes in Computer Science*, pages 36–62. Springer, 2004.
8. Rudolf Freund, Marian Kogler, and Marion Oswald. A general framework for regulated rewriting based on the applicability of rules. In Jozef Kelemen and Alica Kelemenová, editors, *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Paun on the Occasion of His 60th Birthday*, volume 6610 of *Lecture Notes in Computer Science*, pages 35–53. Springer, 2011.
9. Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) P systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer Berlin Heidelberg, 2007.
10. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
11. Sergiu Ivanov. A formal framework for clock-free networks of cells. *International Journal of Computer Mathematics*, 90(4):776–788, 2013.
12. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
13. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, 3 volumes*. Springer, New York, NY, USA, 1997.
14. Dragoş Sburlan. Clock-free P systems. In *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, Italy, June 2004*, pages 372–383, Milano, Italy, June 2004.
15. Bosheng Song, Mario J. Pérez-Jiménez, and Linqiang Pan. An efficient time-free solution to SAT problem by P systems with proteins on membranes. *J. Comput. Syst. Sci.*, 82(6):1090–1099, 2016.
16. Bosheng Song, Tao Song, and Linqiang Pan. Time-free solution to SAT problem by P systems with active membranes and standard cell division rules. *Natural Computing*, 14(4):673–681, 2015.
17. Bosheng Song, Tao Song, and Linqiang Pan. A time-free uniform solution to subset sum problem by tissue P systems with cell division. *Mathematical Structures in Computer Science*, 27(1):17–32, 2017.
18. Tao Song, Luis F. Macías-Ramos, Linqiang Pan, and Mario J. Pérez-Jiménez. Time-free solution to SAT problem using P systems with active membranes. *Theor. Comput. Sci.*, 529:61–68, 2014.
19. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
20. The P Systems Website. `http://ppage.psystems.eu/`.

Membrane Systems and Time Petri Nets

Bogdan Aman[1], Péter Battyányi[2], Gabriel Ciobanu[1], György Vaszil[2]

[1] Romanian Academy, Institute of Computer Science
Blvd. Carol I no. 8, 700505 Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro
[2] Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
battyanyi.peter@inf.unideb.hu, vaszil.gyorgy@inf.unideb.hu

Summary. We investigate the relationship of time Petri nets and different variants of membrane systems. First we show that the added feature of "time" in time Petri nets makes it possible to simulate the maximal parallel rule application of membrane systems without introducing maximal parallelism to the Petri net semantics, then we define local time P systems and explore how time Petri nets and the computations of local time P systems can be related.

1 Introduction

There has been several models applied for describing concurrency, communication and synchronization. Two of them are the graph-based model, known later as Petri nets, developed by C. A. Petri [11] and the tree-like model of embedded membranes called membrane or P systems invented by Gh. Păun [9].

Petri nets are state/transition systems: places are often used to contain information representing conditions in the system being modeled while transitions are used to represent events that can occur to modify the conditions. Some of the information, the input of the transition, is required for an event to happen, while some other information, the output of the transition, provides the result of the executed transition: they are the output of the transition. A Petri net is a bipartite graph: arcs point from input places to transitions and from transitions to places storing their outputs.

There may be some situations where the modelling of a system by conditions and events is not completely satisfactory, for example, when the assumption that all the transitions can take place in an arbitrary order does not describe the system correctly. To model the situation when time delay must be taken into account time Petri nets (TPN) were developed. Concerning time Peri nets, several models were elaborated: time was associated with transitions, places or arcs, etc. We consider the approach adopted by Merlin [8] rendering time to transitions. By this

model, to every transition t we associate a closed interval $[a_t, b_t]$ such that a_t, $b_t \in \mathbb{Q}_{\geq 0}$. The transition can fire, if it is enabled and its local time $h(t)$ is such that $a_t \leq h(t) \leq b_t$. We adopt the strong semantics, which means that a transition which is enabled either must be fired at some point of the associated interval or it becomes non enabled by firing of another transition. In general, time Petri nets are more powerful than ordinary Petri nets, since time Petri nets are able to simulate Turing machines while, for ordinary Petri nets, this is not possible.

Membrane systems are parallel, distributed, synchronized models of computation where embedded membranes are organized in a tree like structure and computation takes place simultaneously in the different membranes in the forms of applications of rewriting rules. The rules evolve in a distributed manner: the application of a rule yields elements with labels, so called messages, which prescribe the exact place where the result of the rule application should move to. An element obtained by a rule application can either remain in the actual membrane, permeate to the parent membrane, or enter into one of its child membranes indicated by the rule. We consider here the basic model, that is, a membrane structure without dissolution rules. In addition we associate to each rule a time interval which gives a lower and an upper values for the time instance when the rule can be executed. We found technically simpler to consider every compartment as if a local stopwatch would operate in that compartment, though the same results could be obtained when we defined a global clock for synchronizing computational steps in the whole membrane system. We call our membrane systems local time membrane systems.

In this paper we relate local time membrane systems to time Petri nets such that the image Petri net of a membrane system by this mapping is suitable for answering questions in connection with the membrane system. For example, we can heavily lean on results in the area of time Petri nets concerning questions of reachability, which asks whether a certain configuration of the membrane system can be achieved, or threshold problems, where the question is whether a state can be reached from another in a certain time, or simply finding the paths requiring minimum/ maximum time between any two reachable states (see Popova-Zeugmann [14]).

There are several timed models for P systems in the literature (see [3], [4], [1]). The attempts for the simulation, up to the present, seem to take the approach similar to timed Petri nets ([6], [1], [2]), where certain values, the delay values, are assigned to rules. This means that the result of a rule application can appear only after that delay assuming a global clock synchronizes the computation of the system. Our model resembles much to that of time Petri nets: an interval is assigned to every rule and a clock local to each compartment synchronizes when the rule can be executed. A computational step is governed by a global clock: only when all membranes finish their action can a new step take place.

2 Membrane systems

First of all, we discuss some terminology used in the sequel. A finite multiset over an alphabet V is a mapping $M : V \to \mathbb{N}$ where $\mathbb{N}$ is the set of non-negative integers, and $M(a)$ for $a \in V$ is said to be the multiplicity of a in V. We say that $M_1 \subseteq M_2$ if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union or sum of two multisets over V is defined as $(M_1 + M_2)(a) = M_1(a) + M_2(a)$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$. The multiset M can also be represented by any permutation of a string $w = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$, where if $M(x) \neq 0$, then there exists j, $1 \leq j \leq n$, such that $x = a_j$. The set of all finite multisets over an alphabet V is denoted by $\mathcal{M}(V)$, the empty multiset is denoted by $\emptyset$ as in the case of the empty set.

A membrane system, or P system, is a tree-like structure of hierarchically arranged membranes embedded in the *skin* membrane as the outermost part of the system. Each region is delimited by a surrounding membrane, they can be arranged in a tree (cell-like [9]) structure or in a graph form (tissue-like [7] or neural-like [5]). In this paper we use the so-called symbol-object P systems [9] without dissolution, that is, each membrane has a label and enclosing a region containing a multiset of objects and rules and possibly some other membranes. The unique outer-most membrane is called the skin membrane. We assume the membranes are labelled by natural numbers $\{1, \dots, n\}$, and we use the notation m_i for the membrane with label i. Each membrane m_i, except for the skin membrane, has its parent membrane, which we denote by $\mu(m_i)$. As an abuse of notation we use μ both for the parent function and both for denoting the structure of the membrane system itself.

The contents of the regions of a P system evolve through rules associated with the regions. The computation of a P system is a locally asynchronous globally synchronous process: each multiset of objects in a region is formed locally by the rules attached to the regions, while a computational step of the whole system is a macro step: it finishes when all of the regions have finished their actions. In the variant we consider in this paper, the rules are multiset rewriting rules given in the form of $u \to v$ where u, v are multisets, and they are applied in a maximal parallel manner, that is, a region finishes its computation when no more rules can be applied in that computational step. In fact, the computational steps in the regions consist of two parts: first the rule application part and then comes a communication part where all the objects with labels find their correct places. The end of the computation of the system is defined by the following halting condition: a P system halts when no more rules can be applied in any of the regions; the result is a number, or a tuple of natural numbers- the number of certain objects in a membrane labelled as output.

Definition 1. *A* P system *of degree* $n \geq 1$ *is* $\Pi = (O, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$ *where*

- *O is an alphabet of objects,*

- μ *is a membrane structure of* n *membranes,*
- $w_i \in \mathcal{M}(O),\ 1 \leq i \leq n$, *are the initial contents of the* n *regions,*
- $R_i,\ 1 \leq i \leq n$, *are the sets of evolution rules associated with the regions; they are of the form* $u \to v$ *where* $u \in \mathcal{M}(O)$ *and* $v \in \mathcal{M}(O \times tar)$ *where* $tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq n\}$.

Unless otherwise stated we consider the n-th membrane as the output membrane. A configuration is the sequence $W = (w_1, \ldots, w_n)$ where w_k are the multiset contents of membrane m_k ($1 \leq k \leq n$). Let $\mathcal{R} = R_1 \cup R_2 \cup \cdots \cup R_n$, where $R_i = \{r_{i1}, \ldots, r_{ik_i}\}$ is the set of rules corresponding to membrane m_i. The application of $u \to v \in R_i$ in the region i means to remove the objects of u from w_i and to add the new objects specified by v to the system. The rule application in each region takes place in a non-deterministic and maximally parallel manner. This means that the rule application phase finishes, if no rule can be applied anymore in any region. As a result, each region where rule applications took place, is possibly supplied with elements of the set $O \times tar$. We call a configuration which is a multiset over $O \cup O \times tar$ an intermediate configuration. If we want to emphasize that $W = (w_1, \ldots, w_n)$ consists of multisets over O, we say that W is a proper configuration. Rule applications can be preceded by priority check, if priority relations are present. Let $\rho_i \subseteq R_i \times R_i$ $1 \leq i \leq n$ be the (possibly empty) priority relations. Then $r \in R_i$ is applicable only if no $r' \in R_i$ can be applied with $(r', r) \in \rho_i$. We may also denote the relation $(r', r) \in \rho_i$ by $r' > r$. Priority relations will be mentioned only in Remark 2.

In the next phase the elements coming from the right hand sides of the rules of region i should be added to the regions as specified by the target indicators associated with them. If $rhs(r)$ contains a pair $(a, here) \in V \times tar$, then a remains in region i, this is the region where the rule is applied. If $rhs(r)$ contains $(a, out) \in V \times tar$, then a is added to the parent region of region i. In our membrane systems we assume that the results are formed in a designated membrane, the output membrane, of the system. Unless otherwise stated, we consider m_n as the output membrane of the system. If $rhs(r)$ contains $(a, in_j) \in V \times tar$ for some region j, then a is added to the contents of region j. In the latter case $\mu(m_j) = m_i$ holds.

3 The Petri net model

By defining a time dependent Petri net model we followed the definition proposed by Popova-Zeugmann [12] and chose a model rendering time intervals to transitions along the original concept of Merlin [8]. First of all, we define the notion of untimed Petri net and then extend this concept to the timed version.

Definition 2. *A Petri net is a tuple* $U = (P, T, F, V, m_0)$ *such that*

1. P, T, F *are finite, where* $P \cap T = \emptyset$, $P \cup T \neq \emptyset$ *and* $F \subseteq (P \times T) \cup (T \times P)$,
2. $V : F \to \mathbb{N}_{>0}$,
3. $m_0 : P \to \mathbb{N}$.

The elements of P are called places and the elements of T are called transitions. The elements of F are the arcs and F is the flow relation of U. The function V is the multiplicity (weight) of the arcs and m_0 is the initial marking. We may occasionally omit the initial marking and simply refer to a Petri net as the tuple $U = (P, T, F, V)$. We stipulate that, for every transition t, there is a place p such that $V(p, t) \neq 0$.

In general, a marking is a function $m : P \to \mathbb{N}$. Let $x \in P$ or $x \in T$. The pre- and postsets of x, denoted by ${}^\bullet x$ and $x^\bullet$, respectively, are defined as ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$. Each arc has an incoming and outcoming multiplicity denoted as follows:

Definition 3. *Let t be a transition. We define below two markings, t^- and t^+, as multisets of places, which govern when a transition can be fired and how many tokens are added to the place p upon firing the transition, respectively.*

$$t^-(p) = \begin{cases} V(p,t), & \text{if } (p,t) \in F, \\ 0 & \text{otherwise ,} \end{cases} \qquad t^+(p) = \begin{cases} V(t,p), & \text{if } (t,p) \in F, \\ 0 & \text{otherwise .} \end{cases}$$

A transition is said to be enabled, if $t^-(p) \leq m(p)$ for all $p \in P$. Applying the notation $\triangle\ t = t^+ - t^-$, we are able to define a firing of the Petri net $U = (P, T, F, V)$.

Definition 4. *Let $U = (P, T, F, V, m_0)$ be a Petri net and let m be a marking in U. A transition $t \in T$ can fire in m (notation: $m \longrightarrow^t$), if t is enabled in m. After the firing of t, the Petri net will obtain the new marking m', where*

$$m' = m + \triangle\ t.$$

Notation: $m \longrightarrow^t m'$.

We obtain time Petri nets, if we add to the Petri net model information about time attached to transitions. Intuitively, the time associated to a transition will denote the last time when the transition or a transition with common preplace was fired. Though the definitions could be extended to unbounded time intervals also, we are concerned with bounded time intervals this time.

Definition 5. *A time Petri net (TPN) is a 6-tuple $N = (P, T, F, V, m_0, I)$ such that*

1. *the 5-tuple $S(N) = (P, T, F, V, m_0)$ is a Petri net,*
2. *$I : T \to \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0}$ and, for each $t \in T$, $I(t)_1 \leq I(t)_2$ holds, where $I(t) = [I(t)_1, I(t)_2]$.*

We call $I(t)_1$ and $I(t)_2$ earliest and latest firing times belonging to t, respectively. Notation: $eft(t)$, $lft(t)$.

A function $m : P \to \mathbb{N}$ is called a p-marking of N. Observe that talking about a p-marking of N is the same as talking about a marking of $S(N)$, where $S(N)$ is called the skeleton of N and, roughly speaking, it is the untimed Petri net obtained from N by omitting every reference to time.

Definition 6. *1. A transition marking (or t-marking) is a function* $h : T \to \mathbb{R}_{\geq 0} \cup \{\#\}$.

2. Let $N = (P, T, F, V, m_o, I)$ *be a time Petri net,* m *a p-marking and* h *a t-marking in* N. *A state in* N *is a pair* $u := (m, h)$ *such that*

a) $(\forall t \in T)(t^- \not\leq m \to h(t) = \#)$,

b) $(\forall t \in T)(t^- \leq m \to h(t) \in \mathbb{R}_{\geq 0} \wedge h(t) \leq lft(t))$.

The initial state is the pair $u_0 = (m_0, h_0)$, where m_0 is the initial marking and

$$h_0(t) = \begin{cases} 0, & \text{if } t^- \leq m_0, \\ \# & \text{otherwise .} \end{cases}$$

Definition 7. *A transition* t *is ready to fire in state* $u = (m, h)$ *(in notation:* $u \longrightarrow_t$*), if* t *is enabled and* $eft(t) \leq h(t)$.

We define the result of the firing of a transition that is ready to fire.

Definition 8. *Let* t *be a transition and* $u = (m, h)$ *be a state such that* $u \longrightarrow^t$. *Then the result of the firing of* t *is a new state* $u' = (m', h')$, *such that* $m' = m + \triangle t$ *and*

$$h'(\hat{t}) = \begin{cases} h(\hat{t}), & \text{if } (\hat{t}^- \leq m, \hat{t}^- \leq m' \text{ and } {}^\bullet\hat{t} \cap {}^\bullet t = \emptyset) \text{ or } t = \hat{t}, \\ \# & \text{if } \hat{t}^- \not\leq m', \\ 0 & \text{otherwise .} \end{cases}$$

In words, the firing of a transition has multiple effects. First of all, it changes the t-marking of the system as it is customary by simple Petri nets. Moreover, the time values attached to the transitions may also change. If $\hat{t}$ was enabled before the firing of transition t and $\hat{t}$ remains enabled after the firing, moreover $\hat{t}$ has no common preplace with the transition which has just been fired, then the value $h(\hat{t})$ for $\hat{t}$ remains unchanged. The value $h(\hat{t})$ remains the same even if $\hat{t} = t$. If $\hat{t}$ is newly enabled with the firing of transition t or $\hat{t}$ has common preplace with t and $\hat{t}$ differs from t, then we have $h(\hat{t}) = 0$. If $\hat{t}$ is not enabled after firing of transition t, then $h(\hat{t}) = \#$.

Observe that we adopt a stronger condition for h to preserve the value for a transition $\hat{t}$ upon firing with transition t. We are not content with the fact that $\hat{t}$ should be newly enabled in order to have $h(\hat{t}) = 0$ in the subsequent computational step, but we also demand that t and $\hat{t}$ should not have common preplaces. To ensure multiple executions of the same transition, if $\hat{t} = t$, then $h(\hat{t})$ retains its value after the firing step.

Besides the firing of a transition there is another possibility for a state to alter, and this is the time delay step.

Definition 9. *Let* t *be a transition and* $u = (m, h)$ *be a state and* $\tau \in \mathbb{R}^+$. *Then elapsing of time with* τ *is possible for the state* u *(in notation:* $u \longrightarrow^\tau$*), if for all* $t \in T$, $h(t) \neq \#$ *implies* $h(t) + \tau \leq lft(t)$. *Then the result of the elapsing of time by* τ *is defined as follows:* $u \longrightarrow^\tau u' = (m', h')$, *where* $m = m'$ *and*

$$h'(\hat{t}) = \begin{cases} h(\hat{t}) + \tau, & \text{if } \hat{t}^- \leq m' \text{ for an arbitrary } \hat{t} \in T, \\ \# & \text{otherwise.} \end{cases}$$

Observe that the definition of the result of a time elapse ensures that we are not able to skip a transition when it is enabled: a transition cannot be made not enabled by a time jump. Finally, we define the notion of a feasible run in a time Petri net.

Definition 10. *Let $N = (P, T, F, V, m_o, I)$ be a time Petri net, assume $\sigma = t_1 \dots t_n$ is a sequence of transitions and $\tau = \tau_0 \tau_1 \dots \tau_n$ ($\tau_i \in \mathbb{R}_{\geq 0}$) be a sequence of times. Then $\sigma(\tau)\tau_0 t_1 \tau_1 \dots t_n \tau_n$ is called a run. $\sigma(\tau)$ is a feasible run, if there are states $s = (m, h)$ and $s' = (m', h')$ such that $s \longrightarrow^*_{\sigma(\tau)} s'$. We may omit the argument τ from $\sigma(\tau)$ if it is clear from the context.*

Obviously, classic Petri nets can be obtained when $h(t) = [0, 0]$ for every transition and no time delay step is ever made.

4 Relating the Petri net model to the membrane system

First, we show how to establish a correspondence between the P system model without time and the model of time Petri nets. As the first step we give the underlying structure of the Petri net associated to a membrane system. The correspondence described below seems to have appeared first by Kleijn, Koutny and Rozenberg [6]. They define the correspondence by limiting the results of the Petri net computations only to those which can be obtained by a sequence of maximal parallel or maximally enabled transition steps. A step is a multiset of transitions and a transition is maximally enabled, if it is enabled and is not a proper subset of an enabled step. They establish a close correspondence between Petri nets with maximally enabled (max enabled) steps and membrane systems. Moreover, other semantics like locally enabled steps or minimal enabled steps could be considered.

In this case we preserve the original semantics for Petri nets: the fireable transitions can be executed in any order. This involves that we have to make essential use of the timed model, since ordinary Petri net model is not Turing complete in contrast to the general membrane system.

Definition 11. *Let $\Pi = (O, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$ be a membrane system. Then we define the following places and transitions for the Petri net.*

1. *$P = P_0 \cup P_0^* \cup \{st_o, st_e, sem\}$, where $P_0 = V \times \{1, \dots, k\}$ and $P_0^* = V^* \times \{1, \dots, k\}$. We set $m_0(p) = w_j(a)$ for every place $p = (a, j)$. Intuitively, the places $V \times \{1, \dots, k\}$ correspond to the objects of V labelled by the indexes of the membranes and the places in $V^* \times \{1, \dots, k\}$ correspond to the objects on the right hand sides of R_i ($1 \leq i \leq n$) labelled by messages. The places st_e, st_o, sem are additional places which serve for the synchronization of the Petri net model.*
2. *$T = T_0 \cup T_0^* \cup \{t_o, t_e, t^1_{sem}, t^2_{sem}\}$, where the sets of transitions T_0 and T_0^* are detailed in the subsequent parts of the definition and $\{t_o, t_e, t^1_{sem}, t^2_{sem}\}$ are auxiliary transitions to be specified later. Let $r_l \in R_i$, where $l \in \{1, \dots, n_{k_i}\}$.*

Then let t_l^i *denote the transition corresponding to* r_l *and* $T_0 = \{t_l^i \mid 1 \leq i \leq n, 1 \leq l \leq k_i\}$. *A transition* t_l^i *connects elements of* P_0 *to* P_0^*: *if* $p = (a, j)$, *then* $V(p, t_l^i) = lhs(r_l)(a)$, *if* $i = j$, *and* $V(p, t_l^i) = 0$ *otherwise. Furthermore, if* $p^* = (a^*, j)$, $V(t_l^i, p^*) = rhs(r_l)(a)$, *if* $i = j$, $V(t_l^i, p^*) = rhs(r_l)(a, out)$, *if* $j = parent(i)$ *and* $V(t_l^i, p^*) = rhs(r_l)(a, in_j)$, *if* $i = parent(j)$ *and* $V(t_l^i, p^*) = 0$ *otherwise. Likewise,* $T_0^* = \{s_j^i \mid 1 \leq i \leq k, 1 \leq j \leq n\}$ *are such that* $\{^\bullet(s_j^i) \mid 1 \leq i \leq k, 1 \leq j \leq n\} = P_0^*$, $\{(s_j^i)^\bullet \mid 1 \leq i \leq k, 1 \leq j \leq n\} = P_0$ *and, if* $a_i \in O$ *and* $1 \leq j \leq n$, *then* $p^* = (a_i, j)^*$ *and* $V(p^*, s_j^i) = V(s_j^i, (a_i, j)) = 1$ *and all the other values are 0. The transitions* t_{sem}^1 *and* t_{sem}^2, *belonging to the semaphore, will be treated in the section.*

3. *The intervals belonging to the elements of* $T = T_0 \cup T_0^*$ *are* $[0, 0]$, *the transitions aiming for the synchronization have various time intervals to be specified later.*

In words, we simulate the rule $r_l \in R_i$ with transition t_l^i such that the weights of the arcs reflect the multiplicities of the elements in compartment i, and the transitions s_j^k ensure the correct reordering of the elements with messages when the rewriting phase is finished. If we term the rule application phase as the odd and the communication phase as the even part of the operation, we obtain two Petri nets for the subsequent phases of the simulation, which are illustrated in Figures 1 and 2. A little more detailed, the complete Petri net acts as follows. The two main sets for the places correspond to the objects of the membrane system. If $a_i \in V$ has k_i occurrences in m_j, then, for $p = (a_i, j)$, $m(p) = k_i$. Likewise, assume at the end of a rule application phase we have k_i' occurrences of $(a_i, here)$ in m_j, and k_i'' occurrences of (a_i, out) in m_l, where $j = parent(l)$ and k_i''' copies of (a_i, in_j) for $l = parent(j)$, then $m(p^*) = k_i' + k_i'' + k_i'''$, where $p^* = (a_i, j)^*$. At the rule application phase the element st_o controls the process: if there are any transitions that are enabled, then they are executed. Otherwise a time elapse is applied and a token from st_o is passed over to *sem* at time 1. The situation is similar with the communication phase: if every element has found its correct place, then no more transition s_i^j is possible and st_e gives control to *sem* by passing a token to *sem* at time instance 1.

We ensure the globally asynchronous locally synchronous character of the membrane system for the Petri net by defining a semaphore which governs the distinct groups of membrane transitions, like rewriting phase, where objects are replaced in accordance with rewriting rules, or communication phase, where objects labelled with tags in_j, *here, out* find their correct places.

In what follows we define the timed part of the Petri net that provides the synchronization.

Assume the semaphore is denoted by the tuple $Sem = (\{sem\}, R, F_{sem}, V_{sem}, I)$. In some sense the semaphore divides the rule application and communication parts of the operation of the P system. The place *sem* of the semaphore is the place where this choice takes place. The place *sem* obtains either one or two tokens.

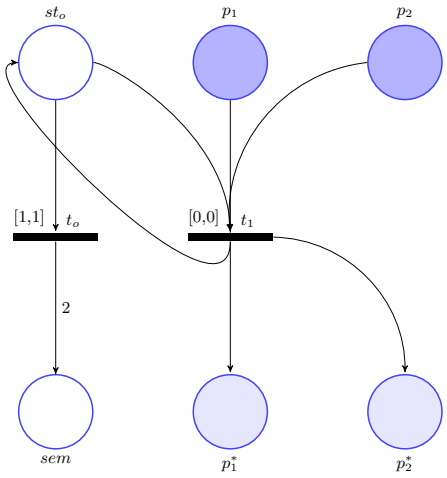


Fig. 1. The Petri net simulating the rule application part of a membrane computational step.

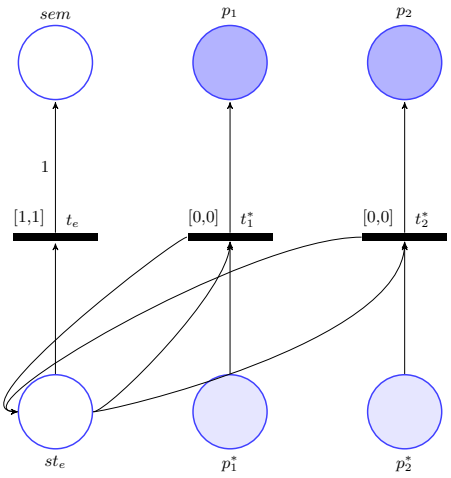


Fig. 2. The Petri net simulating the communication part of a membrane computational step.

If the odd phase is finished, then the semaphore obtains 2 tokens and otherwise, from the even phase, it obtains 1. One of the transitions require 2 tokens with time interval $[0, 0]$- this transitions leads to st_e, and the other one requires 1 token with time interval $[0, 0]$. The latter transition points to st_o. This means that two tokens enable the semaphore to activate the even phase, on the other hand, if at the end of the even phase it receives back only one token, then, after a time jump of 1, only the odd phase can be activated. We illustrate the semaphore in Figure 3.

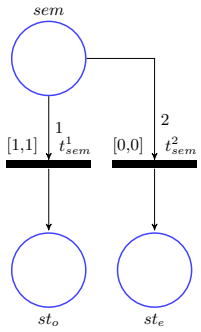


Fig. 3. The semaphore for the Petri nets.

By this, we have simulated a membrane system with a time Petri net such that in the Petri net model no restriction on the transitions is made: every transition which is ready to fire can be fired in any order. We summarize the above considerations in the following theorem.

Theorem 1. *Let $\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ be a membrane system. Then there exists a time Petri net $N = (P, T, F, V, m_0, I)$ defined as in Definition 11 such that, for any computational sequence W of Π yielding an output, there exists a feasible run of N yielding the same output as W.*

We remark that, by keeping the core construction, it is not difficult to adjust the Petri net model so that it is able to simulate membrane systems defined with semantics other than the maximal parallel semantics. As a short remark, we consider lmax-parallelism (or locally max-enabledness) that was treated by Koutny and Kleijn and Rozenberg [6]. A computational step is lmax-parallel in a membrane system, if, for every membrane, the rules of the compartment are executed in a maximally parallel manner, or no rule of that membrane is executed at all in that computational step. To handle lmax-parallelism, Koutny and Kleijn and Rozenberg introduced localities for a Petri net. We define the notion of Petri nets with localities in accordance with the definition of Koutny and Kleijn and Rozenberg [6].

Definition 12. *A Petri net with localities (in short PNL) is a tuple* $NL = (P, T, V, D, m_0)$*, where* P*,* T*,* V *and* m_0 *are as defined by the base model and* $D : T \to \mathbb{N}$ *is a locality mapping. As in the original model, we assume that, for every transition* t *there is a place* p *such that* $V(p,t) \neq 0$*.*

The reason for introducing localities in [6] was the intention of simulating computation distributed to compartments in a membrane system. In the language of Petri nets with localities this is achieved when we define a computational step U as a multiset of transitions so that $l \in D(U)$ implies that U is maximal with respect to the transitions with $D(t) = l$. In our timed model we achieve lmax parallelism by inserting a nondeterministic choice at the beginning of every odd round: either a maximal parallel computational step is simulated concerning region l, or no computation takes place with respect to that region in the underlying computational step. We omit the details.

As a final remark, we stress out that our simulating Petri net works in a one-step manner in contrast to the model defined by Kleijn, Koutny and Rozenberg [6]. This means that exploring the state space seems to be an easier task by this model- there is no need to maintain a huge stack for keeping track of the possible successors of the present state obtained by a maximal parallel step. Hence, from the practical point of view using a time Petri net model for a membrane system seems to be promising.

5 Local time membrane systems

In this section we associate time to the rules of P systems and present a translation from local time P systems into time Petri nets such that the durations of the membrane computational steps can be estimated with the elapsed time in the computation of the time Petri net. This allows us to formulate several properties of local time membrane systems based on the well developed theory of time Petri nets. In the first part of the section we formulate the necessary definitions what we mean by local time membrane systems and elapsed time in a local time membrane system, then we present the simulation of membrane systems by Petri nets.

Definition 13. *We define the notion of a local time P system as a P system together with functions* $\mathcal{I} : R \to IntQ$ *and* $\mathcal{T} : \{1, \ldots, n\} \to \mathbb{R}_{\geq 0}$ $(1 \leq i \leq n)$*, where* R *is the set of all rules and* $IntQ$ *is the set of all closed intervals with nonnegative rational endpoints and there are* n *compartments of* Π*. The value* $\mathcal{T}(i)$ *is called the local time for the* i*-th membrane. A configuration of a local time P system is a pair* $(W, \mathcal{T})$*, where* W *is the configuration as a multiset of objects and messages as before and* $\mathcal{T}$ *is stands for the local time function. We may write* $\mathcal{T}_i$ *for* $\mathcal{T}(i)$ *in the sequel.*

Observe that, since $IntQ$ denotes the intervals with rational endpoints, we may assume that the endpoints of the intervals belonging to the rules are integers.

We have to multiply with the least common multiplier of the nominators of the endpoints in any other case. Likewise for the case of the time Petri nets. Next we define a computational step in a local time P system.

Definition 14. *Let $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time P system. A run is a tuple $\sigma = (\sigma_1, \ldots, \sigma_n)$, where $\sigma_i = \tau_1^i r_1^i \ldots \tau_{k_i}^i r_{k_i}^i \tau_{k_i+1}^i$ such that $\tau_j^i \in \mathbb{R}_{\geq 0}$ and $r_j^i \in R_i$ for $1 \leq j \leq k_i$ and $1 \leq i \leq n$. We call the element σ_i the i-th selection. Moreover, we stipulate that the rules in σ_i form a maximal parallel multiset of rules. The elapsed time for σ_i is the sum of the τ-s in the selection.*

Remark 1. It seems to us that instead of defining local times for each compartment separately we could have chosen to give a global clock for the whole membrane system when talking about a computational step. The definition with a global clock could be given in such a way that not only the evolving of the system but every quantitative property, like elapsed time during a computational step, minimal/maximal time between two configurations, would remain the same. Technically, it seems to be a clearer formulation to introduce a local clock in each membrane, though.

Next we describe how the system can evolve during a selection belonging to a membrane. To facilitate the treatment we shall talk about the configuration of membrane i, as well, not only about a configuration of the whole system.

Definition 15. *Let m_i be a compartment with (possible) intermediate configuration w_i. Then $(w_i, \mathcal{T}_i)$ is the (timed) configuration of m_i.*

In what follows, when we talk about a configuration of a local time membrane system, we mean a timed configuration of the system, unless otherwise stated. If we want to emphasize that we talk about configurations without the time component, then we will use the term object configuration. The computation in a compartment can evolve through two steps when we consider a selection.

Definition 16. *1. rule execution: Assume $r \in R_i$, for some $1 \leq i \leq n$, is enabled, that is, $lhs(r) \leq w_i$, where $(w_i, \mathcal{T}_i)$ is the configuration for m_i. Moreover, assume r is ready to be executed, which means r is enabled and $\mathcal{T}_i \in [\mathcal{I}(r)^-, \mathcal{I}(r)^+]$. Then $(w_i, \mathcal{T}_i) \longrightarrow_r (w_i', \mathcal{T}_i)$, where $w_i' = w_i - lhs(r) + rsh(r)$.*
2. time elapse: Let $\tau \in \mathbb{R}_{\geq 0}$. Then we distinguish two types of semantics:
a) weak semantics: $(w_i, \mathcal{T}_i) \longrightarrow_r (w_i, \mathcal{T}_i')$ and $\mathcal{T}_i' = \mathcal{T}_i + \tau$,
b) strong semantics: $(w_i, \mathcal{T}_i) \longrightarrow_r (w_i, \mathcal{T}_i')$ and $\mathcal{T}_i' = \mathcal{T}_i + \tau$ only if, for every $r \in R_i$, $lhs(r) \leq w_i$ and $\mathcal{T}_i \leq \mathcal{I}(r)^+$ implies $\mathcal{T}_i + \tau \leq \mathcal{I}(r)^+$.

A configuration $(w, \mathcal{T})$ can evolve in two ways. If a rule $r \in R_i$ is enabled and, furthermore, the local time for m_i is such that $\mathcal{T}_i$ lies in the interval $[\mathcal{I}(r)^-, \mathcal{I}(r)^+]$, then r can be executed. Rule execution in a membrane does not change the time part of a configuration, it changes only the multiset of objects in the underlying membrane. The second possibility is time elapse. If we adopt the weak semantics,

then the time elapse step can take place at any time instance. This may involve that a rule which was ready to be executed before the time elapse step can be no more executed after that step. When we adopt the strong semantics this situation is impossible. In other words, the strong semantics does not allow us to skip rules that are enabled and their time interval makes their application possible by a time jump. The weak semantics permits us to apply an arbitrary choice of the rules. As usual, computations in the membranes evolve by a maximal parallel manner. When every membrane has finished its operation, the communication phase of the computational step begins, as by the non-timed case. The new computational step starts with a configuration $(W, \mathcal{T}_0)$, where $\mathcal{T}_0(i) = 0$ $(1 \leq i \leq n)$. By a computation, we understand a sequence of subsequent proper configurations, that is, configurations with no objects labelled by messages. When the computation halts, the result is stored in the output membrane, as before.

6 Local time P systems and time Petri nets

Next we give simulations for local time membrane systems where the elapsed time is the sum of the time jumps in a selection and the elapsed time for a run or a computational step is the maximum of these sums when we consider the different compartments. We perform the simulation for both the weak and strong semantics of local time membrane systems.

First of all, we deal with the case of a membrane system with the weak semantics. The underlying model is the same as in Definition 11, and we try to simulate membrane systems of the weak semantics by time Peri nets equipped with the strong semantics. We identify rule applications with transitions as before. In order to keep the local nature of the Petri net, we split our model into subnets corresponding to the execution in the various compartments. The main idea is that we divide the possible time duration of the computation in a membrane membrane into time intervals of length 1, and, concerning these time intervals, a nondeterministic choice of the transitions which are ready to fire is considered. The transitions in the same group are assumed to take place in the time interval $[0, 0]$, $[0, 1]$ or $[1, 1]$, a group of transitions is initiated by a place q_j^i as illustrated in Figure 4. When operating with a group of transitions, we may apply time elapse between firing steps up to the point when we fall into the next group of transitions. The passage is provided by transitions n_j^i in Figure 4. Let us apply the following notation in the definition below. Assume B is the least integer greater than the right hand sides of the intervals $\mathcal{I}(r)$. Let m_i be a membrane. Let

$$R_j^i = \{r \in R_i \mid j \in [eft(r), lft(r)]\}$$

for $0 \leq j \leq B-1$, $j \in \mathbb{N}$ and $1 \leq i \leq n$. Then $\cup_{i=1}^{n} \cup_{j=0}^{B-1} R_j^i = R$. Observe that the distinct sets R_j^i may overlap for different j-s.

Definition 17. *Let $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system with the weak semantics. Assume B is the least integer greater than the right hand sides of the intervals $\mathcal{I}(r)$. We define the associated time Petri net as follows.*

1. *The main constituents of P are the sets P_0, P_0^*, Q_i $(1 \leq i \leq n)$, where $P_0 = V \times \{1, \ldots, n\}$, $P_0^* = V^* \times \{1, \ldots, n\}$ are as in Definition 11. The computational step is governed by the set of places $Q_i = \{q_j^i \mid 0 \leq j \leq B, 1 \leq i \leq n\}$. There are some additional places including the places for the semaphore, places to mark the end of the computation in the simulated compartment, and places which lead to nonterminating computations. The semaphore is the same as in the previous models. Moreover, since the computational process in every compartment evolves separately, this is modelled by the Petri net: there are sub Petri nets the computations in which are triggered in a distributive manner. The operation of the sub Petri net corresponding to membrane m_i is initiated by transferring a token to the places q_0^i. At the end of the operation of the sub Petri net the place fin_i obtains a token and waits for the rest of the sub nets simulating the other compartments to finish computing. As usual, P_0 represents the actual configuration of the membrane system. We set $m_0(p) = w_j(a)$ for every place $p = (a, j)$. As before, the places in $V \times \{1, \ldots, n\}$ correspond to the objects on the left hand sides of R_i $(1 \leq i \leq n)$ of the membrane rules, while the elements of $V^* \times \{1, \ldots, n\}$ correspond to the objects on the right hand sides of the membrane rules labelled by messages.*
2. *As before, T_0 correspond to the membrane rules, while the transitions of T_0^* ensure the simulation of the communication phase of a membrane computational step. Let $r_l \in R_i$, where $l \in \{1, \ldots, n_{k_i}\}$. Then T_0 formed by t_l^i $(1 \leq i \leq n, 1 \leq l \leq k_i)$ and $T_0^* = \{s_j^i \mid 1 \leq i \leq k, 1 \leq j \leq n\}$ are defined as in Definition 11 with the slight modification as follows. Let $r \in R_j^i$, assume t is the transition associated with r in T_0. Then $(q_j^i, t) \in F$, $V(q_j^i, t) = 1$ and $(t, q_j^i) \in F$, $V(t, q_j^i) = 1$ $(1 \leq j \leq B-1, 1 \leq i \leq n)$. Moreover, for each each q_j^i $(1 \leq j \leq B-1)$ we have transitions n_j^i and t_{Bj}^i such that $(q_j^i, n_j^i) \in F$, $(n_j^i, q_{j+1}^i) \in F$ and $(q_j^i, t_{Bj}^i) \in F$, $(t_{Bj}^i, q_B^i) \in F$ with multiplicities 1. Furthermore, every transition in $t \in T_0$ corresponding to some $r \in R_i$ has a second copy $\widetilde{T_0}$, which is connected to q_B^i by $(q_B^i, \tilde{t}) \in F$ and, for every $p \in P_0$, we have an arrow pointing to $\tilde{t}$ if and only if $(p, t) \in F$ with the same multiplicity as that of (p, t). There is a state $perp_{\tilde{t}}$ for every $\tilde{t}$ which induces an infinitely working sub Petri-net. Finally, there is an arrow from q_B^i pointing to fin_i through transition t_{fin_i}. The places fin_i lead to sem through a transition fin_o. When we consider the case of T_0^*, it is enough to apply one auxiliary place, say fin_e, to check whether every transition in T_0^* has finished its operation: $(fin_e, \tilde{t}) \in F$, $(\tilde{t}, fin_e) \in F$ and $(fin_e, t_{fin_e}) \in F$, $(t_{fin_e}, sem) \in F$.*
3. *As to the timings, let $r \in R_j^i$, assume t corresponds to r in the Petri net. If $j = lft(r)$ then let $\mathcal{I}(t) = [0, 0]$, and if $[j, j+1] \subseteq [eft(r), lft(r)]$, then $\mathcal{I}(t) = [0, 1]$, else, if $j = eft(r)$, then $\mathcal{I}(t) = [1, 1]$. Let the members of T_0^**

have intervals $[0,0]$. *For the other transitions:* $\mathcal{I}(n_j^i) = [0,1]$, $\mathcal{I}(\tilde{t}_j) = [0,0]$ *and* $\mathcal{I}(t_{fin_i}) = [1,1]$. *The semaphore is the same as in the core model.*

Figure 4 presents a snapshot of the model: it illustrates the Petri net assigned to membrane m_i.

Fact 2 *Let* $r \in R^i_{j_{min}}, \ldots, r \in R^i_{j_{max}}$, *let* $j_{min} \leq j \leq j_{max}$, *assume* $[i_t^-, i_t^+]$ *is the interval belonging to* t *in the sense of Definition 17 when considering* $r \in R^i_j$, *where* t *is assigned to* r. *Then* $\mathcal{I}(r)^- \leq i_t^- + j \leq i_t^+ + j \leq \mathcal{I}(r)^+$. *Moreover,* $i_t^- + j_{min} \leq \mathcal{I}(r)^-$ *and* $\mathcal{I}(r)^+ \leq i_t^+ + j_{max}$.

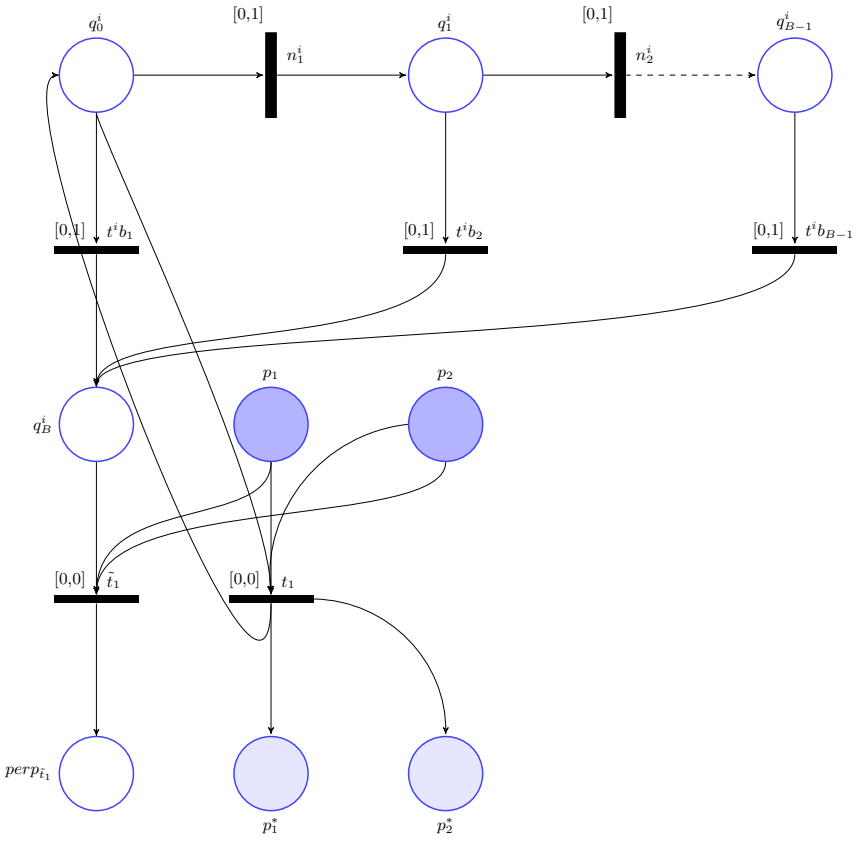


Fig. 4. The Petri net for a local time membrane system with the weak semantics

As a continuation of Figure 4 we show the role q_B^i plays in deciding whether the simulation of the computational step can be continued or not. When the Petri net finishes the simulation of a rule application phase it must check whether the

rules corresponding to the chosen transitions form a maximal parallel set of rules. If at the end of the rule application phase at least one rule remains that could be applied, then our choice is obviously not a maximal parallel set of rules. In order to ensure the correct simulation, in this case the Petri net enters into an infinite loop of transitions when reaching state $perp_{\tilde{t}}$ provided $\tilde{t}$ could be applied. Otherwise, control is given back to the semaphore. Figure 5 details the Petri net which is in fact a sub net of the one in Figure 4.

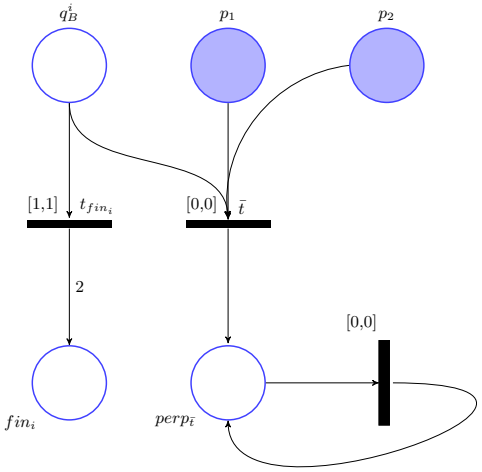


Fig. 5. The Petri net deciding whether a maximal parallel set of rules is reached.

Finally, Figure 6 gives an overall picture of the odd part of the simulating Petri net. The place st_o stimulates the odd phase and tokens are immediately distributed among the places q_0^i, which initiate the simulations of the computations in membranes m_i $(1 \leq i \leq n)$. When the computation in m_i is over, in the simulating Petri net the place fin_i obtains a token. If all the places fin_i $(1 \leq i \leq n)$ have collected their tokens, then a token is passed over to sem and a new computational phase begins.

We remark that the construction above gives a general method for simulating any time Petri net defined with the weak semantics by a time Petri net understood with the strong semantics.

The case for the membrane system with the strong semantics is possibly a bit simpler, since it is closer to the original semantics of the Petri net model. We define in the next definition the simulating Petri net for a local time membrane system with the strong semantics. The construction is very similar to the ones of the previous subsection, probably it is a little bit easier this time. The only difficulty is to tell when a maximal parallel step is finished. For this purpose, before choosing

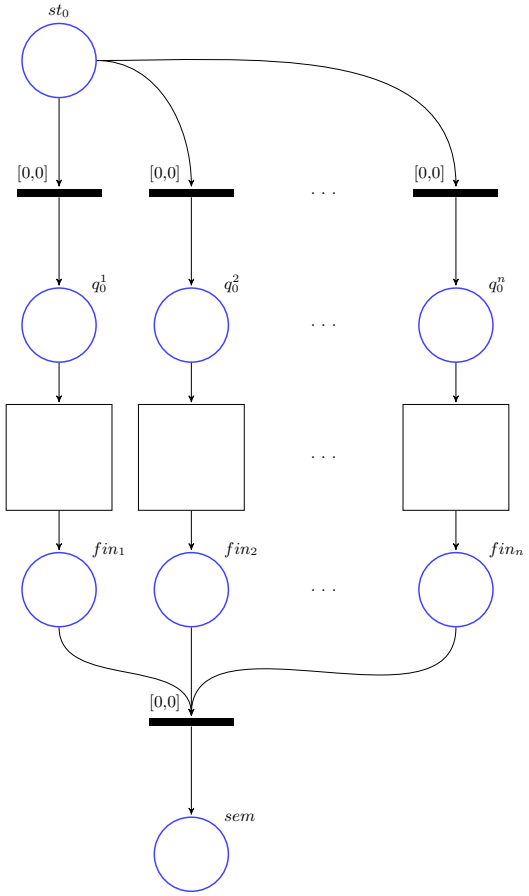


Fig. 6. The overall structure of the Petri net for the weak semantics

the next transition, we implement a test whether there are transitions that could be executed. This involves creating a new copy of the Petri net simulating the left hand sides of the membrane system rules. Moreover, to keep ourselves to the interpretation of the membrane computational step by distinguishing the computational sequences in the different compartments, we assume that there are n sub Petri nets modelling the computations in the different membranes. We detach, as usual, the odd and even phases of the computation. In the odd phase, the token in st_o is immediately distributed to the places st_o^i hence initializing the computation in the sub Petri nets corresponding to membranes m_i.

Definition 18. *Let* $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ *be a local time membrane system with the strong semantics.*

1. *P contains P_0 and P_0^*, where $P_0 = V \times \{1, \ldots, n\}$, $P_0^* = V^* \times \{1, \ldots, n\}$, where $a \in V$ and $(1 \leq j \leq n)$ are in Definition 11. As usual, P_0 represents the actual configuration of the membrane system. We set $m_0(p) = w_j(a)$ for every place $p = (a, j)$. As before, the places in $V \times \{1, \ldots, n\}$ correspond to the objects on the left hand sides of R_i $(1 \leq i \leq n)$ of the membrane rules, while the elements of $V^* \times \{1, \ldots, n\}$ correspond to the objects on the right hand sides of the membrane rules labelled by messages. There are auxiliary places, namely those of the semaphore and st_e, st_o^i, cat_i: they govern the simulation of the rule application and communication phases.*
2. *T consists of T_0, T_0^*, $\widetilde{T}_0$ and some auxiliary transitions. As before, let $r_l \in R_i$, where $l \in \{1, \ldots, n_{k_i}\}$. Then let t_l^i denote the transition corresponding to r_l and $T_0 = \{t_l^i \mid 1 \leq i \leq n, 1 \leq l \leq k_i\}$. A transition t_l^i connects elements of P_0 to P_0^*: if $p = (a, j)$, then $V(p, t_l^i) = lhs(r_l)(a)$, if $i = j$, and $V(p, t_l^i) = 0$ otherwise. Furthermore, if $p^* = (a_i, j)^*$, then $V(t_l^i, p^*) = rhs(r_l)(a)$, if $i = j$, $V(t_l^i, p^*) = rhs(r_l)(a, out)$, if $i = parent(j)$ and $V(t_l^i, p^*) = rhs(r_l)(a, in_j)$, if $j = parent(i)$ and $V(t_l^i, p^*) = 0$ otherwise. Moreover, (cat_i, t_l^i), (t_l^i, st_o^i), (st_o^i, t_{cat_i}), $(t_{cat_i}, cat_i) \in F$.*
   $T_0^* = \{s_j^i \mid 1 \leq i \leq k, 1 \leq j \leq n\}$ is defined as before: we let $\{{}^\bullet(s_j^i) 1 \leq i \leq k, 1 \leq j \leq n\} \mid = P_0^*$, $\{(s_j^i)^\bullet \mid 1 \leq i \leq k, 1 \leq j \leq n\} = P_0$ and $V((p_i^*, j), s_j^i) = V(s_j^i, (a_i, j)) = 1$, where $p_i^* = (a_i, j)^*$, and all the other values be 0.
   As to the auxiliary places and transitions, st_o^i and st_e have to check whether there are transitions left to fire. For st_e this is easy: st_e connects to each transition in T_0^*, that is, (st_e, s_l^i), $(s_l^i, st_e) \in F$ and st_e connects to sem by (st_e, tr_e), $(tr_e, sem) \in F$. The places st_o^i need to make a similar check concerning maximal parallel execution: $(st_o^i, \tilde{t}_l^i) \in F$, $(\tilde{t}_l^i, cat_i) \in F$ and $\tilde{t}_l^i$ are such that the connections with the elements of P_0 are the same as in the case of P_0 and T_0 with the same multiplicities, as well. However, $\tilde{t}_l^i$ do not point to P_0^*, they give back all the tokens to P_0 after any of the transitions $\tilde{t}_l^i$ has been fired. When no transition $\tilde{t}_l^i$ is able to fire, st_o^i forwards a token to fin_i and, when each fin_i $(1 \leq i \leq n)$ possesses a token, they give control back to sem by (fin_i, mon_o), $(mon_o, sem) \in F$. That is, mon_o fires only if every sub Petri net assigned to membrane m_i finishes its computation.
3. *Concerning the intervals: the intervals belonging to the elements of T_0^* are $[0, 0]$. If $t_l^i \in T_0$, then $\mathcal{I}(t_l^i) = \mathcal{I}(r_l^i)$. Furthermore, $\mathcal{I}(fin_i) = \mathcal{I}(fin_e) = [1, 1]$. All the remaining intervals are $[0, 0]$. The semaphore is the same as by the case of the core model.*

We illustrate the sub Petri net corresponding to m_i in Figure 7.

Let $\Pi = (V, \mu, u_1, \ldots, u_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system. If Π is considered with the weak semantics, then let $N_w(\Pi)$ denote the time Petri net associated to Π according to Definition 17. If Π is understood with the strong semantics, then let $N_s(\Pi)$ denote the Petri net assigned to Π in accordance with Definition 18. Furthermore, if $(w, \mathcal{T})$ is a proper configuration of Π, let $(\nu(w), \nu(\mathcal{T}))$ be the configuration of $N_w(\Pi)$ or of $N_s(\Pi)$, where

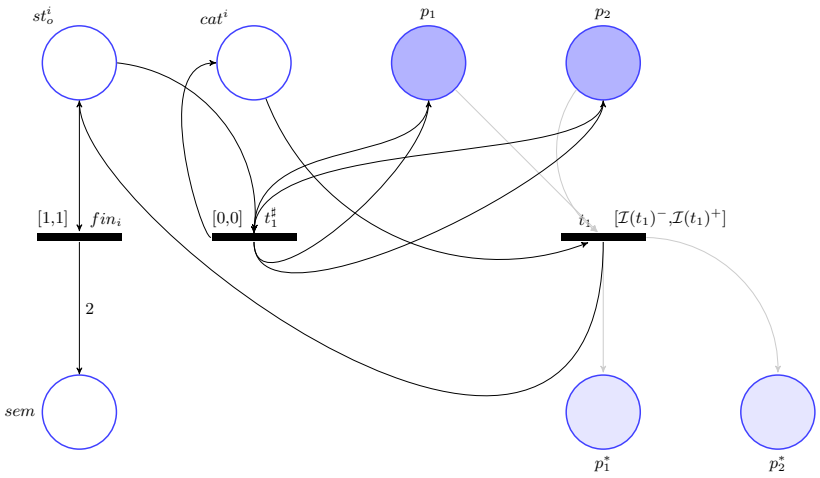


Fig. 7. The Petri net for a local time membrane system with the strong semantics

$\nu(w)(a_i, j) = w(j)(i)$ and $\nu(\mathcal{T})(t_l^i) = \mathcal{T}(r_l)$, where $r_l \in R_i$. In addition, if $p^* = (a_i^*, j)$, let $\nu(w)(p^*) = 0$, as $(w, \mathcal{T})$ is a proper configuration. Moreover, if $W = (w^0, \mathcal{T}^0) \Rightarrow (w^1, \mathcal{T}^1) \Rightarrow \ldots \Rightarrow (w^k, \mathcal{T}^k)$ is a configuration sequence of Π, then $\nu(W) = (\nu(w^0), \nu(\mathcal{T}^0)) \longrightarrow (\nu(w^1), \nu(\mathcal{T}^1)) \longrightarrow \ldots \longrightarrow (\nu(w^k), \nu(\mathcal{T}^k))$ is the corresponding sequence of configurations of $N_w(\Pi)$ or $N_s(\Pi)$. If W is a sequence of proper configurations, then we omit the values $\mathcal{T}_i$ and $\nu(\mathcal{T}_i)$ from the configurations $(w, \mathcal{T})$ and $(\nu(w), \nu(\mathcal{T})_i)$, respectively, since, in this case $\mathcal{T}^j = \mathcal{T}_0$, where $\mathcal{T}_0(r) = 0$ for every $r \in R$.

Theorem 3. *Let $\Pi = (V, \mu, u_1, \ldots, u_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system.*

1. *Let $W = w^0 \Rightarrow w^1 \Rightarrow \ldots \Rightarrow w^k$ be a computational sequence with the weak semantics. Then*

$$w^0 \Rightarrow_W^* w^k \Leftrightarrow N_w(w^0) \longrightarrow_{\nu(W)}^* N_w(w^k).$$

   Moreover, $\tau(\nu(W)) = \tau(W) + 3k$, where $\tau(W)$ and $\tau(\nu(W))$ are the total time for W and $\nu(W)$, respectively.
2. *Let $W = w^0 \Rightarrow w^1 \Rightarrow \ldots \Rightarrow w^k$ be a computational sequence with the strong semantics. Then*

$$w^0 \Rightarrow_W^* w^k \Leftrightarrow N_s(w^0) \longrightarrow_{\nu(W)}^* N_s(w^k).$$

   Moreover, $\tau(\nu(W)) = \tau(W) + 3k$, where $\tau(W)$ and $\tau(\nu(W))$ are the total time for W and $\nu(W)$, respectively.

Proof. We treat the case of the weak semantics: we give a sketch for the proof of the correctness of our simulation. Assume $\Pi = (V, \mu, u_1, \ldots, u_n, R_1, \ldots, R_n, \mathcal{I})$ is a local time membrane system, let $W = w^0 \Rightarrow w^1 \Rightarrow \ldots \Rightarrow w^k$ be a computational sequence with the weak semantics. We prove the theorem by induction on k.

- $k = 0$: there is nothing to prove.
- $k = l + 1$: Assume we have the statement for l, that is, for the computational sequence $W_l = w^0 \Rightarrow^{\sigma_1} w^1 \Rightarrow^{\sigma_2} \ldots \Rightarrow^{\sigma_l} w^l$ there exists $\xi_l = m^0 \Rightarrow m^1 \Rightarrow \ldots \Rightarrow m^l$ such that $\mu^l = \nu(w^l)$ and $\tau(\nu(\xi_l)) = \tau(W_l) + 3 \cdot l$, where, for any proper configuration w of Π, $\nu(w)$ is defined as above. If q is any place not in P_0, then $\nu(q) = 0$, except for sem, where $\nu(w)(sem) = 1$. By the construction of the Petri net $h(t_l^i) = \mathcal{I}(r_l^i)$ also holds. We extend the correspondence ν to the intermediate configurations, as well. The values for the elements of P_0 are defined as before. As to the values of P_0^*: $\nu(w)(a_i^*, j) = w(j)(a_i, here) + w(\mu(j))(a_i, in_j) + \sum_{j=\mu(l)}(a_i, out)$. Let $\sigma_{(i,s)} = \tau_0 r_1 \tau_1 r_2 \ldots r_s \tau_{s+1}$, where $r_1, \ldots, r_s \in R_i$, be a segment of the selection of m_i of length s with the corresponding configurations $((w_1, \mathcal{T}_1), \ldots, (w_s, \mathcal{T}_s))$. Then $\xi_{(i,s)} = \nu(\sigma_{(i,s)}) = \tau_{01} \ldots \tau_{0k_0} t_1 \tau_{11} \ldots \tau_{1k_1} t_2 \ldots t_s \tau_{(s+1)1} \ldots \tau_{(s+1)k_{s+1}}$, where t_j correspond to r_j in $N_w(\Pi)$ and $\tau_j = \tau_{j1} + \ldots + \tau_{jk_j}$, is a computation in $N_w(\Pi)$. Assume $((m_1, h_1), \ldots, (m_s, h_s))$ is the sequence of states corresponding to $\xi_{i,s}$, then we claim that $\nu(w_1), \ldots, \nu(w_s)$ define exactly the same t-markings. For the correspondence of the configurations and t-markings it is enough to prove that, if r can be executed, then t is ready to fire provided t is assigned to r. By Fact 2, it is enough to observe that $\mathcal{T}_i(r) = j + a$ for some $j \in \mathbb{N}$ such that $r \in R_j^i$ and $0 \leq a \leq 1$, and $i_t^- \leq a \leq i_t^+$, where $[i_t^-, i_t^+]$ is the interval corresponding to t as the image of $r \in R_j^i$ in accordance with Definition 17. The statement can be proved by examining the various cases for the next step in $\sigma_{i,s}$.

□

The converse of the theorem holds, too. We state it in a proposition.

Proposition 1. *Let $N = (P, T, F, V, m_0, I)$ be a time Petri net. Then the following statements are valid.*

1. *Let $\Pi = (V, \mu, u_1, \ldots, u_n, R_1, \ldots, R_n, \mathcal{I})$ be be a local time membrane system, assume $N = N_w(\Pi)$. Then, for any proper computational sequence ξ of N, there exists computational sequence W of Π with respect to the weak semantics such that ξ provides exactly the same output as W. Moreover, $\tau(\xi) = \tau(W) + 3k$, where $\tau(W)$ and $\tau(\xi)$ are the total time for W and ξ, respectively.*
2. *Let $\Pi = (V, \mu, u_1, \ldots, u_n, R_1, \ldots, R_n, \mathcal{I})$ be be a local time membrane system, assume $N = N_w(\Pi)$. Then, for any proper computational sequence ξ of N, there exists computational sequence W of Π with respect to the strong semantics such that ξ provides exactly the same output as W. Moreover, $\tau(\xi) = \tau(W) + 3k$, where $\tau(W)$ and $\tau(\xi)$ are the total time for W and ξ, respectively.*

Remark 2. We remark that local time weak semantics does not appear to add anything to the computational power of the membrane system, however local time with the strong semantics seems to increase the computational strength of the P system. We conjecture that, by a modification of a proof of Păun [10] showing that membrane systems with catalytic rules together with priority define recursively enumerable sets of numbers even with two membranes, it might not be difficult to prove that local time membrane systems with catalytic rules and with the strong semantics define recursively enumerable sets of numbers with two membranes. It is not clear to us, however, how the exact strength of a local time membrane system with the strong semantics could be depicted.

7 Applications

The constructions of the previous section makes us possible to apply the results elaborated for time Petri nets for the case of local time membrane systems.

Notation 4 *Let $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system. We apply the notation $\Pi = \Pi_\star$, where $\star \in \{w, s\}$ stands for either the weak semantics or the strong semantics, respectively.*

Definition 19. *Let $\Pi_\star = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system.*

1. *A (possibly intermediate) configuration $(w, \mathcal{T})$ is integer valued, if $\mathcal{T}(r) \in \mathbb{N}$ for every $r \in R$.*
2. *Let m_i be a membrane of $\Pi_\star$ for some $1 \leq i \leq n$. Let σ_i be a run for m_i. Then σ_i is integer valued if all of its intermediate configurations are integer valued.*
3. *Let $W = w_0 \Rightarrow \ldots \Rightarrow w_k$ be a computational sequence for $\Pi_\star$. Then W is integer valued, if, for every w_i, every run of w_i is integer valued.*

Observe that, given a membrane system $\Pi_\star$ and a computational sequence W, the condition that W is integer valued is equivalent to the requirement that all the time elapses in every run of W are integers.

Proposition 2. *Let $\Pi_\star = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system and let w' be a configuration reachable from w. Then w' is integer reachable from w.*

Proof. Follows from the main theorem together with the corresponding theorem for time Petri nets presented by Popova-Zeugmann ([13], [14]). □

Definition 20. *Let $\Pi_\star = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system. Then $\Pi_\star$ is bounded, if there exists $K > 0$ such that, for every configuration w of $\Pi_\star$, $|w(j)| < K$ $(1 \leq j \leq n)$. In other words, K is an upper bound for the number of elements in every compartment with regard to any configuration w.*

Proposition 3. *Let $\Pi_\star = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system, assume $\Pi_\star$ is bounded. Then the reachability problem for $\Pi_\star$ is decidable.*

Proof. Follows from Proposition 2. □

Proposition 4. *Let $\Pi_\star = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, \mathcal{I})$ be a local time membrane system, assume w' is reachable from w. Then the minimum and maximum distances between w and w' are integers.*

8 Conclusions

In this paper we examined the connections between two models of computations, namely, the membrane systems introduced by Păun [9] and time Petri nets defined along the lines of the papers of Popova-Zeugmann ([12],[13]). First of all, we presented a simulation of membrane systems without dissolution by time Petri nets. The novelty in our result is the fact that the simulating Petri nets manage to retain the locality of firing: transitions can be fired one after the other without structural control, like maximal parallelism, imposed on the order of their execution. Next, we defined local time membrane systems on the analogy of time Petri nets, equipping the computational model with two types of semantics. We showed that both kinds of local time membrane systems can be simulated by time Petri nets with the strong semantics. Finally, we mentioned some statements concerning local time membrane systems that are straightforward consequences of the similar results for time Petri nets by reason of the simulations.

References

1. B. Aman, G. Ciobanu, Adding Lifetime to Objects and Membranes in P Systems. *International Journal of Computers, Communications and Control*, 5(3) (2010) 268–279.
2. B. Aman, G. Ciobanu, Verification of membrane systems with delays via Petri nets with delays. *Theoretical Computer Science*, 598(C) (2015) 87–101.
3. M. Cavaliere, D. Sburlan, Time and Synchronization in Membrane Systems. *Fundamenta Informaticae*, 64(1) (2005) 65–77.
4. M. Cavaliere, D. Sburlan, Timeindependent P Systems Towards a Petri Net Semantics for Membrane Systems. *Lecture Notes in Computer Science*, volume 3365, International Workshop on Membrane Computing, WMC 2004, 239–258, Springer Verlag, Berlin, 2005.
5. M. Ionescu, Gh. Păun, T. Yokomori. Spiking Neural P Systems. *Fundamenta Informaticae*, 71 (2006) 279–308.
6. J. H. C. M. Kleijn and M. Koutny and G. Rozenberg, Towards a Petri Net Semantics for Membrane Systems. *Lecture Notes in Computer Science*, volume 3850, International Workshop on Membrane Computing, WMC 2005, 292–309, Springer Verlag, Berlin, 2005.

7. C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón. Tissue P Systems. *Theoretical Computer Science*, 296 (2003) 295–326.
8. P. M. Merlin, A Study of the Recoverability of Computing Systems. *PhD thesis*, University of California, Irvine, CA, 1974
9. G. Păun, Computing with Membranes. *Journal of Computer and System Sciences*, 61(1) (2000) 108–143.
10. G. Păun, *Membrane Computing - An Introduction*, Springer Verlag, Berlin, 2002.
11. C. A. Petri, Kommunikation mit Automaten. *Dissertation*, Universität Hamburg, Hamburg, 1962.
12. L. Popova, On time Petri nets. *Journal of Information Processing and Cybernetics*, EIK, 27(4) (1991) 227–244.
13. L. Popova-Zeugmann, Essential States in time Petri nets, *Informatik- Berichte der HUB*, Nr. 96, 1998.
14. L. Popova-Zeugmann, *Time and Petri Nets*, Springer Verlag, Berlin, 2013.

Two Notes on APCol Systems

Lucie Ciencialová[1,2] and Luděk Cienciala[1,2]

[1] Research Institute of the IT4Innovations Centre of Excellence,
Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic
[2] Institute of Computer Science
Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic
lucie.ciancialova@fpf.slu.cz ludek.cienciala@fpf.slu.cz

Summary. In this work, we continue our research in the field of string processing membrane systems - APCol systems. We focus on a relation of APCol systems with PM colonies - colonies whose agents can only perform point mutation transformations of the common string, in a vicinity of the agent. The second part is devoted to a connection of APCol systems and logic circuits using AND, OR and NOT gates.

1 Introduction

There are many different theoretical computational models, where are independent agents engage in a shared environment or interact with it directly. In this paper, we continue our research in the examination of connections between such models. In [26] our research started with the comparison of eco-colonies – composed from components (grammars) acting in a string, the string is also evolved by environmental rules (0L scheme), P colonies and eco-P colonies – membrane systems with one-membrane agents and a unordered environment. In this paper, we continue our study with APCol systems – the model related closely to P colonies – PM colonies and logic circuits.

APCol systems are formal models of a computing device combining properties of membrane systems and colonies - parallel distributed systems of formal grammars. Membrane systems (called P systems, [24]) are biologically inspired distributed multiset rewriting systems that are characterised by massive parallelism and simple rules. APCol systems were introduced in [4] The reader can find more information about membrane systems in [23] and in [17] can a reader find details on grammar systems theory.

An APCol system is formed from agents - collections of objects embedded in a membrane - and shared environment - string. Agents are equipped with programs composed of rules that allow agents to interact with objects that are placed inside them and they form a string. The number of objects inside each agent is set by definition and it is usually very low up to 3. The string is processed by agents

and it is used as a communication channel for agents too. Through the string the agents are able to affect the behaviour of another agent. There is a special object defined in the APCol system it is denoted by e. It has a special role: whenever it is introduced in the string, the corresponding input symbol is erased.

The activity of agents is based on rules that can be rewriting, communication or checking; these three types was introduced in [18]. Rewriting rule $a \to b$ allow agent to rewrite (evolve) one object a to object b. Both objects are placed inside the agent. Communication rule $a \leftrightarrow b$ gives to the agent the possibility to exchange object c placed inside the agent for object d from the string. A checking rule is formed from two rules r_1, r_2 of type rewriting or communication. It sets a kind of priority between rules r_1, r_2. The agent tries to apply the first rule and if it cannot be performed, the agent executes the second rule. The rules are combined into programs in such a way that all object are affected by the execution of the rules. Consequently, the number of rules in the program is the same as the number of objects inside the agent.

The computation in APCol systems starts with an input string, representing the environment, and with each agents having only symbols e in its state. Every computational step means a maximally parallel action of the active agents: an agent is active if it is able to perform at least one of its programs, and the joint action of the agents is maximally parallel if no more active agent can be added to the synchronously acting agents. The computation ends if the input string is reduced to the empty word, there are no more applicable programs in the system, and meantime at least one of the agents is in so-called final state.

We start the paper with the necessary definitions not only of APCol systems, PM-colonies and logic circuits. We continue with the construction of an APCol system simulating a PM-colony and with the construction of APCol system suitable for simulation of logic circuits. We conclude the paper with final remarks about future work and open problems.

2 Definitions

Throughout the paper the reader is assumed to be familiar with the basics of formal language theory and membrane computing. For further details we refer to [16] and [23].

For an alphabet Σ, the set of all words over Σ (including the empty word, ε), is denoted by Σ^*. We denote the length of a word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$. For a language $L \subseteq \Sigma^*$, the set $length(L) = \{|w| \mid w \in L\}$ is called the length set of L. For a family of languages FL, the family of length sets of languages in FL is denoted by NFL.

A multiset of objects M is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : V \to N$; f assigns to each object in V its multiplicity in M. The set of all multisets with the set of objects V is denoted by V°. The set V' is called the support of M and denoted by $supp(M)$.

The cardinality of M, denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects $V' = \{a_1, \ldots a_n\}$ can be represented as a string w over alphabet V' with $|w|_{a_i} = f(a_i);\ 1 \leq i \leq\ n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M, and ε represents the empty multiset.

2.1 APCol Systems

In the following we recall the concept of an APCol system [4].

As in the case of standard P colonies, agents of APCol systems contain objects, each being an element of a finite alphabet. With every agent, a set of programs is associated. There are two types of rules in the programs. The first one, called an evolution rule, is of the form $a \ \rightarrow b$. It means that object a inside of the agent is rewritten (evolved) to the object b. The second type of rules, called a communication rule, is in the form $c \leftrightarrow d$. When this rule is performed, the object c inside the agent and a symbol d in the string are exchanged, so, we can say that the agent rewrites symbol d to symbol c in the input string. If $c = e$, then the agent erases d from the input string and if $d = e$, symbol c is inserted into the string.

An APCol system works successfully, if it is able to reduce the given string to ε, i.e., to enter a configuration where at least one agent is in accepting state and the processed string is the empty word.

Definition 1. *[4] An APCol system is a construct*
$\Pi = (O, e, A_1, \ldots, A_n)$*, where*

- O *is an alphabet; its elements are called the objects,*
- $e \in O$*, called the basic object,*
- $A_i,\ 1 \leq i \leq n$*, are agents. Each agent is a triplet* $A_i = (\omega_i, P_i, F_i)$*, where*
  - ω_i *is a multiset over* O*, describing the initial state (content) of the agent,* $|\omega_i| = 2$*,*
  - $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ *is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:*
    - $a \rightarrow b$*, where* $a, b \in O$*, called an evolution rule,*
    - $c \leftrightarrow d$*, where* $c, d \in O$*, called a communication rule,*
  - $F_i \subseteq O^*$ *is a finite set of final states (contents) of agent* A_i*.*

During the work of the APCol system, the agents perform programs. Since both rules in a program can be communication rules, an agent can work with two objects in the string in one step of the computation. In the case of program $\langle a \leftrightarrow b; c \leftrightarrow d \rangle$, a sub-string bd of the input string is replaced by string ac. If the program is of the form $\langle c \leftrightarrow d; a \leftrightarrow b \rangle$, then a sub-string db of the input string is replaced by string ca. That is, the agent can act only in one place in a computation step and the change of the string depends both on the order of the rules in the

program and on the interacting objects. In particular, we have the following types of programs with two communication rules:

- $\langle a \leftrightarrow b; c \leftrightarrow e\rangle$ - b in the string is replaced by ac,
- $\langle c \leftrightarrow e; a \leftrightarrow b\rangle$ - b in the string is replaced by ca,
- $\langle a \leftrightarrow e; c \leftrightarrow e\rangle$ - ac is inserted in a non-deterministically chosen place in the string,
- $\langle e \leftrightarrow b; e \leftrightarrow d\rangle$ - bd is erased from the string,
- $\langle e \leftrightarrow d; e \leftrightarrow b\rangle$ - db is erased from the string,
- $\langle e \leftrightarrow e; e \leftrightarrow d\rangle$; $\langle e \leftrightarrow e; c \leftrightarrow d\rangle$, ...- these programs can be replaced by programs of type $\langle e \rightarrow e; c \leftrightarrow d\rangle$.

The program is said to be *restricted* if it is formed from one rewriting and one communication rule. The APCol system is restricted if all the programs the agents have are restricted.

At the beginning of the work of the APCol system (at the beginning of the computation), there is an input string placed in the environment, more precisely, the environment is given by a string ω of objects which are different from e. This string represents the initial state of the environment. Consequently, an initial configuration of the automaton-like P colony is an $(n+1)$-tuple $c = (\omega; \omega_1, \ldots, \omega_n)$ where ω is the initial state of the environment and the other n components are multisets of strings of objects, given in the form of strings, the initial states the of agents.

A configuration of an APCoL system Π is given by $(w; w_1, \ldots, w_n)$, where $|w_i| = 2,\ 1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent and $w \in (O - \{e\})^*$ is the string to be processed.

At each step of the computation every agent attempts to find one of its programs to use. If the number of applicable programs is higher than one, the agent non-deterministically chooses one of them. At every step of computation, the maximal possible number of agents have to perform a program.

By applying programs, the automaton-like P colony passes from one configuration to another configuration. A sequence of configurations starting from the initial configuration is called a computation. A configuration is halting if the APCol system has no applicable program.

The result of computation depends on the mode in which the APCol system works. In the case of accepting mode, a computation is called accepting if and only if at least one agent is in final state and the string obtained is ε. Hence, the string ω is accepted by the automaton-like P colony Π if there exists a computation by Π such that it starts in the initial configuration $(\omega; \omega_1, \ldots, \omega_n)$ and the computation ends by halting in the configuration $(\varepsilon; w_1, \ldots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

The situation is different when the APCol system works in the generating mode. A computation is called successful if only if it is halting and at least one agent is in final state. The string w_F is generated by Π iff there exists computation starting in an initial configuration $(\varepsilon; \omega_1, \ldots, \omega_n)$ and the computation ends by halting in the configuration $(w_F; w_1, \ldots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

We denote by $APCol_{acc}R(n)$ (or $APCol_{acc}(n)$) the family of languages accepted by APCol system having at most n agents with restricted programs only (or without this restriction). Similarly we denote by $APCol_{gen}R(n)$ the family of languages generated by APCol systems having at most n agents with restricted programs only.

APCol system Π can generate or accept set of numbers $|L(\Pi)|$.

By $NAPCol_xR(n), x \in \{acc, gen\}$, the family of sets of natural numbers accepted or generated by APCol systems with at most n agents is denoted.

In [4] the authors proved that the family of languages accepted by jumping finite automata (introduced in [21]) is properly included in the family of languages accepted by APCol systems with one agent, and it is proved that any recursively enumerable language can be obtained as a projection of a language accepted by an automaton-like P colony with two agents.

In [4] the reader can find following theorems about accepting power of APCol systems:

- The family of languages accepted by APCol system with one agent properly includes the family of languages accepted by jumping finite automata.
- Any recursively enumerable language can be obtained as a projection of a language accepted by an APCol system with two agents.

The results about generative power of APCol systems are shown in [2]:

- Restricted APCol systems with only two agents working in generating mode can accept any recursively set of natural numbers. $NAPCol_{gen}R(2) = NRE$
- A family of sets of natural numbers acceptable by partially blind register machine can be generated by an APCol system with one agent with restricted programs. $NRM_{PB} \subseteq NAPCol_{gen}R(1)$

2.2 PM-colonies

In this part we recall the definition of PM-colonies that was introduced in [20].

Definition 2. *A PM-colony (of degree $n; n \geq 1$) is a construct*

$$\pi = (E; \#; N; >; R_1; \ldots; R_n);$$

where E is an alphabet (of the environment), $\#$ is a special symbol not in E (the boundary marker of the environment), $N = \{A_1; \ldots; A_n\}$ is the alphabet of agents names, $>$ is a partial order relation over N (the priority relation among agents), and $R_1; \ldots; R_n$ are finite sets of action rules of the agents. The action rules can be of the following forms:

$$
\begin{array}{lll}
\textit{Deletion:} & (a;A_i;b) \to (\varepsilon;A_i;b), & \textit{for } a \in E \cup N; b \in E \cup N \cup \{\#\},\\
& (a;A_i;b) \to (a;A_i;\varepsilon), & \textit{for } a \in E \cup N \cup \{\#\}; b \in E \cup N,\\
\textit{Insertion:} & (a;A_i;b) \to (a;c;A_i;b), & \textit{for } a;b \in E \cup N \cup \{\#\}; c \in E \cup N,\\
& (a;A_i;b) \to (a;A_i;c;b), & \textit{for } a;b \in E \cup N \cup \{\#\}; c \in E \cup N,\\
\textit{Substitution:} & (a;A_i;b) \to (c;A_i;b), & \textit{for } b \in E \cup N \cup \{\#\}; a;c \in E,\\
& (a;A_i;b) \to (a;A_i;c), & \textit{for } a \in E \cup N \cup \{\#\}; b;c \in E,\\
\textit{Move:} & (a;A_i;b) \to (A_i;a;b), & \textit{for } a \in E \cup N; b \in E \cup N \cup \{\#\},\\
& (a;A_i;b) \to (a;b;A_i), & \textit{for } a \in E \cup N \cup \{\#\}; b \in E \cup N,\\
\textit{Death:} & (a;A_i;b) \to (a;\varepsilon;b), & \textit{for } a;b \in E \cup N \cup \{\#\}.
\end{array}
$$

Each action has a premise of the form $(a;A_i;b)$, it specifies the agent and its neighbouring symbols, and has a consequence where the agent is still present, with only one exception, when it disappears; the boundary marker cannot be changed or overpassed.

The state of the environment is described by a string of the form $\#w\#$, where $w \in (E \cup N)^*$. The string $pr_E(w)$ describes the environment without the agent population, $pr_N(w)$ describes the agent population. One agent can appear in several copies in the environment.

The agents act on the environment symbols as well as on other agents, in a local manner, according to the action rules; the agents can also change their position in the environment, according to the move rules, but they cannot step outside the boundary markers.

Agents in PM-colony work in parallel manner. Similarly as in the other parallel working systems, conflicts can occur between agents. If in a word $w \in (E \cup N)^*$ context overlay of two agents A_i and A_j happens or if agent A_j takes part in context of agent A_i, we call it direct conflict between agents. If in w the pairs of agents $(A_1, A_2), (A_2, A_3), \ldots, (A_n, A_{n+1})$ are in direct conflict then the whole set of agents $A_1, A_2, A_3, \ldots, A_n, A_{n+1}$ are in conflict. The conflict of agents $A_1, A_2, A_3, \ldots, A_n, A_{n+1}$ in PM-colony can be solved by the agent with the greatest priority, which takes action. So, to solve the conflict, conflicting agents have to be ordered in such a way, that there is an agent with priority higher then all other agents in the conflict. Moreover the agent with the greatest priority occurs in the conflict set only once.

Let A be agent of PM-colony and $\#w\# = xaAby$ be a configuration of PM-colony, where $a, b \in (E \cup N) \cup \{\#\}$. This occurrence of agent A is active with respect to configuration $\#w\#$, if (1) an action rule exists, whose left side is in the form (a, A, b), and (2) A is not conflicting with any other agent occurrence, or A has the highest priority from all agents from those in conflict. An agent occurrence is inactive, if it is not active.

A derivation step in a PM-colony denoted as $\Rightarrow$ is a binary relation on a set of configurations. We write $\#w\# \Rightarrow \#z\#$ if and only if each active agent A in the string w replaces its context in w by corresponding rule and the resultant string is $\#z\#$. Derivation $\Rightarrow^*$ is the reflexive and transitive closure of relation $\Rightarrow$. A language associated with PM-colony $\pi = (E; \#; N; >; R_1; \ldots; R_n)$ and a starting state w_0 of its environment is defined as

$$L(\pi; w_0) = \{x \in E^* \mid w_0 \Rightarrow^* w;\ x = pr_E(w)\}$$

2.3 Logic gates and boolean circuits

Boolean circuits are a formal model of the combinational logic circuits. Circuits consist of wires able to carry one bit, and logical gates connecting such wires and computing the elementary logical functions, NOT, AND, OR. To simplify circuits used in practice, more logical gates were introduced: NOR, XOR and NAND. Value tables and graphical symbols are shown in Table 1.

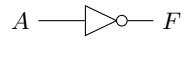
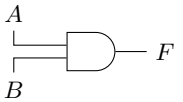
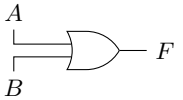
Name of logic gate	Graphical symbol	Value table
NOT		A | F 0 | 1 1 | 0
AND		A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1

Table 1. Graphical symbols and value tables for logic gates

Definition 3. *A Boolean circuit $\alpha = (V, E, \lambda)$ is a finite directed acyclic graph (V, E), with set of vertices V, and set of directed edges E, and $\lambda : V \to \{I\} \cup \{AND, OR, NOT\}$ a vertex labelling, where I is a special symbol. A vertex $x \in V$ with $\lambda(x) = I$ has indegree 0 and is called an input. A vertex $y \in V$ with outdegree 0 is called an output. Vertices with labels AND and OR have indegree 2 and outdegree 1, while vertices with the label NOT have indegree and outdegree 1.*

3 APCol Systems Versus PM-colonies

In this section we construct APCol system simulating derivation steps in PM-colony. The idea is to perform one derivation step of PM-colony in four phases of computation in APCol system. In the first phase one agent rewrite all occurrences of PM-colony agents into new symbols of a type ${}_aA_b$ were a, b is a context of current copy of agent A. In the second phase the agent check whether there is any conflict.

If the answer is positive, it tries to solve collision. The third phase is performing phase, when APCol system agent rewrite active and inactive agents into objects in the way corresponding to executed rules. If there is more than one applicable rule (PM-colony is non-deterministic) executed rule is chosen non-deterministically. In the last phase the APCol system agent non-deterministically chooses whether computation will end and erase all occurrences of PM-colony agents or the system will continue with the next derivation step - to ensure that APCol system can halt after every simulated derivation step of PM-colony.

Theorem 1. *To every PM-colony π there exist APCol system that can generate all states of PM-colony.*

Proof. Let $\pi = (E; \#; N; >; R_1; \ldots; R_n)$ be a PM-colony operating on a string $w_0 \in (E \cup N)^*$. We construct APCol system $\Pi = (O, e, A_1, \ldots, A_m)$, where m equals to number of rules in π plus 6, as follows:

The agent A_1 has initial configuration ee and the subset of programs to initialize simulation.

A_1

1. $\langle e \to \#'; e \to T\rangle$
2. $\langle \#' \leftrightarrow \#; T \leftrightarrow a\rangle$ for all $a \in E \cup N \cup \{\#\}$
3. $\langle \# \to P; a \to a\rangle$
4. $\langle P \leftrightarrow e; a \leftrightarrow T\rangle$ for all $a \in E \cup N \cup \{\#\}$
5. $\langle T \to\uparrow; e \to e\rangle$

APCol system starts computation on the string $\#w_0\# = \#aw_1\#$. Agent A_1 is in the initial configuration ee. In the first step it rewrites its contents to $\#'T$ using the program 1. At the second step The agent replace the first two symbols from the string by its contents. After second step the contents of the agent A_1 $\#a$ and the string in the environment is $\#'Tw_1\#$. In the next step agent rewrites $\#$ to P and exchange its contents by T from a string. The configuration of the agent is eT and the string has a form $\#'Paw_1\#$.

The agent A_2 is supporting agent; It generates the auxiliary objects consumed by agent A_1. Set P_2 contains following programs:

A_2

A. $\langle e \to\downarrow; e \to e\rangle$

B. $\langle \downarrow\leftrightarrow P; e \to e\rangle$

step	string	agent A_1	agent A_2	applicable programs of A_1	applicable programs of A_2
0.	$\#aw_1\#$	ee	ee	1.	A.
1.	$\#aw_1\#$	$\#'T$	$\downarrow e$	2.	$-$
2.	$\#'Tw_1\#$	$\#a$	$\downarrow e$	3.	$-$
3.	$\#'Tw_1\#$	Pa	$\downarrow e$	4.	$-$
4.	$\#'Paw_1\#$	eT	$\downarrow e$	5.	B
5.	$\#' \downarrow aw_1\#$	$\uparrow e$	Pe	6.	C

Phase 1. - Context detection

After symbol $\downarrow$ appears in the string $\#' \downarrow aw_1\#$, agent A_1 goes through the string from the left to the right end and rewrites occurrences of PM-colony agents A by symbols ${}_aA_b$ where a, b are neighbouring symbols (aAB is substring of the input string).

A_1	A_2
6. $\langle \uparrow \leftrightarrow \downarrow; e \leftrightarrow a \rangle$ for all $a \in E \cup N$	$C.\ \langle P \rightarrow L'; e \rightarrow e \rangle$
7. $\langle a \leftrightarrow \uparrow; \downarrow \leftrightarrow e \rangle$ for all $a \in E$	$D.\ \langle L' \leftrightarrow b; e \leftrightarrow L \rangle$ for all $b \in E \cup {}_i\overline{N}_j$ $i, j \in E \cup N \cup \{\#\}$
8. $\langle a \rightarrow \overline{A}; \downarrow \rightarrow L \rangle$ for all $a \in N$	$E.\ \langle b \rightarrow b; L \rightarrow L_b \rangle$
9. $\langle \overline{A} \rightarrow \overline{A}; L \leftrightarrow \uparrow \rangle$	$F.\ \langle b \leftrightarrow L'; L_b \leftrightarrow e \rangle$
	$G.\ \langle L'' \rightarrow R'; e \rightarrow e \rangle$

A_1	A_2
10. $\langle \overline{A} \rightarrow \overline{A}; \uparrow \leftrightarrow L' \rangle$	$H.\ \langle R' \leftrightarrow R; e \leftrightarrow c \rangle$ $c \in E \cup N$
11. $\langle \overline{A} \rightarrow \overline{A}; L' \rightarrow L'' \rangle$	$I.\ \langle R \rightarrow R_c; c \rightarrow c \rangle$
12. $\langle \overline{A} \rightarrow \overline{A}; L'' \leftrightarrow \uparrow \rangle$	$J.\ \langle R_c \leftrightarrow R''; c \leftrightarrow e \rangle$
13. $\langle \overline{A} \rightarrow \overline{A}; \uparrow \leftrightarrow L_{b'} \rangle$	$K.\ \langle R'' \rightarrow L'; e \rightarrow e \rangle$
14. $\langle \overline{A} \rightarrow {}_{b'}A; L_{b'} \rightarrow R \rangle$	
15. $\langle {}_{b'}A \rightarrow {}_{b'}A; R \leftrightarrow \uparrow \rangle$	
16. $\langle {}_{b'}A \rightarrow {}_{b'}A; \uparrow \leftrightarrow R' \rangle$	
17. $\langle {}_{b'}A \rightarrow {}_{b'}A; R' \rightarrow R'' \rangle$	
18. $\langle {}_{b'}A \rightarrow {}_{b'}A; R'' \leftrightarrow \uparrow \rangle$	
19. $\langle {}_{b'}A \rightarrow {}_{b'}A; \uparrow \leftrightarrow R_{c'} \rangle$ where $b, c \in E \cup N$	
20. $\langle {}_{b'}A \rightarrow {}_{b'}A_{c'}; R_{c'} \rightarrow \downarrow \rangle$ where $b, c \in E \cup N$	
21. $\langle {}_{b'}A_{c'} \leftrightarrow e; \downarrow \leftrightarrow \uparrow \rangle$ where $b, c \in E \cup N$	
22. $\langle \downarrow \rightarrow E_x; \# \rightarrow \# \rangle$	
23. $\langle E_x \leftrightarrow \uparrow; \# \leftrightarrow e \rangle$	

Agents A_1 and A_2 helps each other to change all occurrences of PM-colony agents to more complex symbols capturing agent neighbouring symbols. In following table, $a, b \in E$ and $A \in N$.

step	string	agent A_1	agent A_2	applicable programs of A_1	applicable programs of A_2
5.	$\#' \downarrow aAbw_1\#$	$\uparrow e$	Pe	6.	C.
6.	$\#' \uparrow Abw_1\#$	$\downarrow a$	$L'e$	7.	–
7.	$\#'a \downarrow Abw_1\#$	$\uparrow e$	$L'e$	6.	–
8.	$\#'a \uparrow bw_1\#$	$\downarrow A$	$L'e$	8.	–
9.	$\#'a \uparrow bw_1\#$	$\overline{A}L$	$L'e$	9.	–
10.	$\#'aLbw_1\#$	$\overline{A} \uparrow$	$L'e$	–	D.
11.	$\#'L'bw_1\#$	$\overline{A} \uparrow$	aL	10.	E.
12.	$\#' \uparrow bw_1\#$	$\overline{A}L'$	aL_a	11.	–
13.	$\#' \uparrow bw_1\#$	$\overline{A}L''$	aL_a	12.	–
14.	$\#'L''bw_1\#$	$\overline{A} \uparrow$	aL_a	–	F.
15.	$\#'aL_abw_1\#$	$\overline{A} \uparrow$	$L''e$	13.	G.
16.	$\#'a \uparrow bw_1\#$	$\overline{A}L_a$	$R'e$	14.	–
17.	$\#'a \uparrow bw_1\#$	${}_aAR$	$R'e$	15.	–
18.	$\#'aRbw_1\#$	${}_aA \uparrow$	$R'e$	–	H.
19.	$\#'aR'w_1\#$	${}_aA \uparrow$	Rb	16.	I.
20.	$\#'a \uparrow w_1\#$	${}_aAR'$	R_bb	17.	–
21.	$\#'a \uparrow w_1\#$	${}_aAR''$	R_bb	18.	–
step	string	agent A_1	agent A_2	applicable programs of A_1	applicable programs of A_2
22.	$\#'aR''w_1\#$	${}_aA \uparrow$	R_bb	–	J.
23.	$\#'aR_bbw_1\#$	${}_aA \uparrow$	$R''e$	19.	K.
24.	$\#'a \uparrow bw_1\#$	${}_aAR_b$	$L'e$	20.	–
25.	$\#'a \uparrow bw_1\#$	${}_aA_b \downarrow$	$L'e$	21.	–
26.	$\#'a\ {}_aA_b \downarrow bw_1\#$	$\uparrow$	$L'e$	6.	–

Phase 2. - Determining agents as active or inactive

In the second phase agents A_1 and A_3 read the string from right to the left and they search for conflicts. Inactive agents are marked by ${}_y\overline{X}_z$, active agents has the label in a form ${}_yX_z$.

A_1
24. $\langle \uparrow \rightarrow \Uparrow; e \rightarrow \updownarrow \rangle$
25. $\langle \updownarrow \rightarrow \Downarrow; \Uparrow \leftrightarrow \# \rangle$
26. $\langle \Downarrow \leftrightarrow e; \# \leftrightarrow \Uparrow \rangle$

A_3
i. $\langle e \rightarrow \updownarrow; e \rightarrow e \rangle$
ii. $\langle \updownarrow \leftrightarrow E_x; e \rightarrow e \rangle$
iii. $\langle E_x \rightarrow \Updownarrow; e \rightarrow e \rangle$
iv. $\langle \Updownarrow \leftrightarrow \Downarrow; e \rightarrow e \rangle$
v. $\langle \Downarrow \rightarrow e; e \rightarrow e \rangle$

A_1
To skip $a \in E$
27. $\langle \Uparrow \leftrightarrow a; e \leftrightarrow \Updownarrow \rangle$
28. $\langle \Updownarrow \leftrightarrow e; a \leftrightarrow \Uparrow \rangle$

A_1

If A_1 consumes ${}_bA_c, A \in N; b, c \in E \cup N \cup \{\#\}$

29. $\langle \Uparrow\leftrightarrow\ {}_bA_c; e \rightarrow \Updownarrow \rangle$
30. $\langle\ {}_bA_c \rightarrow\ {}_bA'_c; \Updownarrow \rightarrow \Updownarrow_A \rangle$

$$ {}_bA'_c = \begin{cases} {}_bA_c \text{ for } b, c \in E \text{ or } b, c \cup N \text{ and } b < A, c < A \\ {}_b\overline{A}_c \text{ otherwise} \end{cases}$$

31. $\langle \Updownarrow_A \leftrightarrow \Uparrow;\ {}_bA'_c \leftrightarrow e \rangle$

A_1

If $\Updownarrow_A$ is sent to the string and next symbol is $a \in E$

32. $\langle \Uparrow\leftrightarrow a; e \leftrightarrow \Updownarrow_A \rangle$
33. $\langle \Updownarrow_A \rightarrow \Downarrow_A; a \rightarrow a \rangle$
34. $\langle \Downarrow_A \leftrightarrow e; a \leftrightarrow \Uparrow \rangle$

A_1

If $\Updownarrow_A$ is sent to the string and next symbol is ${}_bB_c$

35. $\langle \Uparrow\leftrightarrow\ {}_bB_c; e \leftrightarrow \Updownarrow_A \rangle$
36. $\langle\ {}_bB_c \rightarrow\ {}_bB'_c; \Updownarrow_A \rightarrow Y \rangle$;

$$ {}_bB'_c = \begin{cases} {}_bB_c \text{ for } B > A \text{ and } \{b, c \in E \text{ or } b, c \in N \text{ and } b < A, c < A\} \\ {}_b\overline{B}_c \text{ otherwise} \end{cases}$$

$$ Y = \begin{cases} \Updownarrow_B & \text{for } B > A \\ \Updownarrow_A & \text{for } A > B \\ \Updownarrow_{\{A,B\}} & \text{otherwise} \end{cases}$$

37. $\langle X \leftrightarrow \Uparrow;\ {}_bB'_c \leftrightarrow e \rangle$

A_1

If $\Updownarrow_M$ (M is a subset of N) is sent to the string and next symbol is ${}_bB_c$

35′. $\langle \Uparrow\leftrightarrow\ {}_bB_c; e \leftrightarrow \Updownarrow_M \rangle$
36′. $\langle\ {}_bB_c \rightarrow\ {}_bB'_c; \Updownarrow_M \rightarrow Y \rangle$;

$$ {}_bB'_c = \begin{cases} {}_bB_c \text{ if } B \text{ is greatest element of } M \cup \{B\} \text{ and} \\ \quad b, c \in E \text{ or } b, c \cup N \text{ and} \\ \quad b, c \text{ is not greatest element of } M \cup \{b, c\} \\ {}_b\overline{B}_c \text{ otherwise} \end{cases}$$

$$ Y = \begin{cases} \Updownarrow_B & \text{for } B > A \\ \Updownarrow_A & \text{for } A > B \\ \Updownarrow_{\{A,B\}} & \text{otherwise} \end{cases}$$

37′. $\langle X \leftrightarrow \Uparrow;\ {}_bB'_c \leftrightarrow e \rangle$

A_1

If $\Downarrow_A$ is sent to the string and next symbol is $a \in E$

38. $\langle \Uparrow\leftrightarrow a; e \leftrightarrow \Downarrow_A \rangle$
39. $\langle a \rightarrow a; \Downarrow_A \rightarrow \Updownarrow \rangle$;
40. $\langle \Updownarrow\leftrightarrow\Uparrow; a \leftrightarrow e \rangle$

A_1

If $\Downarrow_A$ is sent to the string and next symbol is ${}_bB_c$

41. $\langle \Uparrow\leftrightarrow\ {}_bB_c; e \leftrightarrow \Downarrow_A \rangle$
42. $\langle\ {}_bB_c \rightarrow\ {}_bB'_c; \Downarrow_A \rightarrow X \rangle$;

$$ {}_bB'_c = \begin{cases} {}_bB_c \text{ for } B > A \text{ and } \{b, c \in E \text{ or } b, c \cup N \text{ and } b < A, c < A\} \\ {}_b\overline{B}_c \text{ otherwise} \end{cases}$$

$$X = \begin{cases} \Updownarrow_A \text{ for } A > B \\ B_A \text{ for } B > A \\ e_A \text{ otherwise} \end{cases}$$

43. $\langle X \leftrightarrow \Uparrow;\ {}_bB'_c \leftrightarrow e \rangle$

If the next symbol to be checked is boundary object $\#'$ the agent A_4 finishes the second phase.

A_5

$AA.\ \langle e \rightarrow D; e \rightarrow e \rangle$
$AB.\ \langle D \leftrightarrow \#'; e \leftrightarrow X \rangle ; X \in \{\Updownarrow; \Updownarrow_A; \Downarrow_A\}$
$AC.\ \langle X \rightarrow C; \#' \rightarrow \#' \rangle$
$AD.\ \langle \#' \leftrightarrow D; C \leftrightarrow e \rangle$

When symbol B_A or symbol e_A appears in the string agent A_5 starts to work by consuming the symbol and replacing it by two symbols: $\Uparrow_A$ for agent A_6 that moves right to set PM-colony agent A as inactive and the second symbol for agent A_1 to continue computation.

A_5

$a.\ \langle e \rightarrow Z; e \rightarrow e \rangle$
$b.\ \langle Z \leftrightarrow B_A; e \rightarrow e \rangle$
$c.\ \langle Z \leftrightarrow e_A; e \rightarrow e \rangle$
$d.\ \langle B_A \rightarrow \Updownarrow_B; e \rightarrow \Uparrow_A \rangle$
$e.\ \langle e_A \rightarrow \Updownarrow; e \rightarrow \Uparrow_A \rangle$
$f.\ \langle \Updownarrow_B \leftrightarrow Z; \Uparrow_A \leftrightarrow e \rangle$
$g.\ \langle \Updownarrow \leftrightarrow Z; \Uparrow_A \leftrightarrow e \rangle$

A_6

$I.\ \langle e \rightarrow B; e \rightarrow e \rangle$
$II.\ \langle B \leftrightarrow \Uparrow_A; e \leftrightarrow X \rangle$;
$\quad X \in E \cup \{\ {}_bB_c \mid b, c \in E \cup N \cup \{\#, \#'\}, B \in N - \{A\}\}$
$III.\ \langle X \leftrightarrow B; \Uparrow_A \leftrightarrow e \rangle$
$IV.\ \langle B \leftrightarrow \Uparrow_A; e \rightarrow\ {}_bA_c \rangle$
$V.\ \langle \Uparrow_A \rightarrow e;\ {}_bA_c \rightarrow\ {}_b\overline{A}_c \rangle$
$VI.\ \langle\ {}_b\overline{A}_c \leftrightarrow B; e \rightarrow e \rangle$

Phase 3. - Simulation of execution of the rules

This phase starts with arrows change. Instead of $\Uparrow$ and $\Updownarrow$ the agent A_1 uses $\uparrow\uparrow$ and $\downarrow\downarrow$. The change is done by agent A_1 and A_5.

A_1	A_5
44. $\langle \Uparrow \leftrightarrow C; e \rightarrow e \rangle$	$h.\ \langle Z \leftrightarrow \#'; e \leftrightarrow \Uparrow \rangle$
45. $\langle C \rightarrow \downdownarrows; e \rightarrow e \rangle$	$i.\ \langle \#' \rightarrow \#'; \Uparrow \rightarrow \uparrow\uparrow \rangle$
46. $\langle \downdownarrows \leftrightarrow \uparrow\uparrow; e \rightarrow e \rangle$	$j.\ \langle \#' \leftrightarrow Z; \uparrow\uparrow \leftrightarrow e \rangle$

The agent A_1 reads the string from the left to the right, it skips symbols from $E \cup N$, changes ${}_b\overline{A}_c$ to A and sends messages to performing agent to add, move, rewrite or delete symbols.

A_1

47. $\langle \uparrow\uparrow \leftrightarrow \downdownarrows; e \leftrightarrow a \rangle \qquad a \in E \cup N$
48. $\langle \uparrow\uparrow \leftrightarrow \downdownarrows; e \rightarrow {}_b\overline{A}_c \rangle$
49. $\langle a \leftrightarrow \uparrow\uparrow; \downdownarrows \leftrightarrow e \rangle$
50. $\langle \downdownarrows \rightarrow \downdownarrows;\ {}_b\overline{A}_c \rightarrow A \rangle$
51. $\langle A \leftrightarrow \uparrow\uparrow; \downdownarrows \leftrightarrow e \rangle$
52. $\langle \uparrow\uparrow \leftrightarrow \downdownarrows; e \rightarrow {}_bA_c \rangle$

Using the program 52 agent A_1 consumes symbol corresponding to active agent. For every rule of PM-colony there exists sets of programs of agent A_1 and one agent performing the action.

If the rule is deletion of the form
$(a; A_i; b) \rightarrow (\varepsilon; A_i; b)$, for $a \in E \cup N; b \in E \cup N \cup \{\#\}$,
$(a; A_i; b) \rightarrow (a; A_i; \varepsilon)$, for $a \in E \cup N \cup \{\#\}; b \in E \cup N$,
the programs are following:

A_1

53. $\langle\ {}_aA_b \rightarrow A; \downdownarrows \rightarrow D_{xy} \rangle$
xy is La or Rb depending on which side the symbol have to be deleted
54. $\langle A \leftrightarrow e; D_{xy} \leftrightarrow \uparrow\uparrow \rangle$

A_{del}

$A1.\ \langle e \rightarrow J; e \rightarrow e \rangle$
$A2.\ \langle J \leftrightarrow D_{xy}; e \rightarrow e \rangle$;
$A3.\ \langle D_{La} \rightarrow d_{La}; e \leftrightarrow \downdownarrows \rangle$
$A4.\ \langle d_{La} \leftrightarrow A; \downdownarrows \leftrightarrow J \rangle$
$A5.\ \langle A \rightarrow A; J \rightarrow J_0 \rangle$
$A6.\ \langle J_0 \leftrightarrow d_{La}; A \leftrightarrow e \rangle$
$A7.\ \langle d_{La} \leftrightarrow a'; e \leftrightarrow J_0 \rangle$ a' is a if $a \in E$ or a' is ${}_d\overline{a}_A$
$A8.\ \langle J_0 \rightarrow J_1; a' \rightarrow e \rangle$
$A9.\ \langle J_1 \rightarrow J_2; e \leftrightarrow d_{La} \rangle$
$A10.\ \langle J_2 \rightarrow J; d_{La} \rightarrow e \rangle$
$A11.\ \langle D_{Rb} \rightarrow d_{Rb}; e \leftrightarrow \downdownarrows \rangle$
$A12.\ \langle \downdownarrows \leftrightarrow J; d_{Rb} \leftrightarrow b' \rangle$ b' is b if $b \in E$ or b' is ${}_A\overline{b}_d$
$A13.\ \langle b' \rightarrow e; J \rightarrow J_0 \rangle$
$A14.\ \langle J_0 \rightarrow J_1; e \leftrightarrow d_{Rb} \rangle$
$A15.\ \langle J_1 \rightarrow J; d_{Rb} \rightarrow e \rangle$

If the rule is insertion of the form
$(a; A_i; b) \rightarrow (a; c; A_i; b)$, for $a; b \in E \cup N \cup \{\#\}; c \in E \cup N$,
$(a; A_i; b) \rightarrow (a; A_i; c; b)$, for $a; b \in E \cup N \cup \{\#\}; c \in E \cup N$,
the programs are following:

A_1

---

55. $\langle\ {}_aA_b \to A; \downdownarrows \to I_{xy}\rangle$

xy is Lc or Rc depending on which side the symbol have to be inserted

56. $\langle A \leftrightarrow e; I_{xy} \leftrightarrow \upuparrows\rangle$

A_{ins}

$A1.\ \langle e \to J; e \to K\rangle$
$A2.\ \langle J \leftrightarrow A; K \leftrightarrow I_{Lc}\rangle$; $A13.\ \langle J \leftrightarrow I_{Rc}; K \to c\rangle$;
$A3.\ \langle I_{Lc} \to I^0_{Lc}; A \to A\rangle$ $A14.\ \langle I_{Rc} \to I^0_{Rc}; c \to c\rangle$
$A4.\ \langle I^0_{Lc} \leftrightarrow J; A \leftrightarrow K\rangle$ $A15.\ \langle c \leftrightarrow J; I^0_{Rc} \leftrightarrow e\rangle$
$A5.\ \langle J \leftrightarrow I^0_{Lc}; K \to c\rangle$ $A16.\ \langle J \leftrightarrow I^0_{Rc}; e \to \downdownarrows\rangle$
$A6.\ \langle I^0_{Lc} \to I^1_{Lc}; c \to c\rangle$ $A17.\ \langle I^0_{Rc} \to K; \downdownarrows \leftrightarrow J\rangle$
$A7.\ \langle c \leftrightarrow J; I^1_{Lc} \leftrightarrow e\rangle$
$A8.\ \langle J \leftrightarrow I^1_{Lc}; e \leftrightarrow A\rangle$
$A9.\ \langle I^1_{Lc} \to I^2_{Lc}; A \to A\rangle$
$A10.\ \langle A \leftrightarrow J; I^2_{Lc} \leftrightarrow e\rangle$
$A11.\ \langle J \leftrightarrow I^2_{Lc}; e \to \downdownarrows\rangle$
$A12.\ \langle I^2_{Lc} \to K; \downdownarrows \leftrightarrow J\rangle$

If the rule is substitution of the form
$(a; A_i; b) \to (c; A_i; b)$, for $b \in E \cup N \cup \{\#\}; a; c \in E$,
$(a; A_i; b) \to (a; A_i; c)$, for $a \in E \cup N \cup \{\#\}; b; c \in E$,
the programs are following:

A_1

---

57. $\langle\ {}_aA_b \to A; \downdownarrows \to S_{xy}\rangle$

xy is Lc or Rc depending on which side the symbol have to be replaced

58. $\langle A \leftrightarrow e; S_{xy} \leftrightarrow \upuparrows\rangle$

A_{subs}

$A1.\ \langle e \to J; e \to e\rangle$
$A2.\ \langle J \leftrightarrow S_{Lc}; e \to e\rangle$; $A11.\ \langle J \leftrightarrow S_{Rc}; e \to e\rangle$;
$A3.\ \langle S_{Lc} \to s_{Lc}; e \to \downdownarrows\rangle$ $A12.\ \langle S_{Rc} \to s_{Rc}; e \to c\rangle$
$A4.\ \langle s_{Lc} \leftrightarrow A; \downdownarrows \leftrightarrow J\rangle$ $A13.\ \langle c \leftrightarrow J; s_{Rc} \leftrightarrow b\rangle$
$A5.\ \langle A \to A; J \to J_0\rangle$ $A14.\ \langle J \to J^0; b \to \downdownarrows\rangle$
$A6.\ \langle J_0 \leftrightarrow a; A \leftrightarrow s_{Lc}\rangle$ $A15.\ \langle J^0 \to J^1; \downdownarrows \leftrightarrow s_{Rc}\rangle$
$A7.\ \langle s_{Lc} \to s^0_{Lc}; a \to c\rangle$ $A16.\ \langle J^1 \to J; s_{Rc} \to e\rangle$
$A8.\ \langle s^0_{Lc} \leftrightarrow J_0; c \leftrightarrow e\rangle$
$A9.\ \langle J^0 \to J^1; e \leftrightarrow s^0_{Lc}\rangle$
$A10.\ \langle J_1 \to J; s^0_{Lc} \to e\rangle$

If the rule is move of the form
$(a; A_i; b) \to (A_i; a; b)$, for $a \in E \cup N; b \in E \cup N \cup \{\#\}$,
$(a; A_i; b) \to (a; b; A_i)$, for $a \in E \cup N \cup \{\#\}; b \in E \cup N$,
the programs are following:

A_1

---

59. $\langle\ {}_aA_b \to A; \downdownarrows \to M_{xy}\rangle$

xy is LA or RA depending on which side the symbol have to be moved

60. $\langle A \leftrightarrow e; M_{xy} \leftrightarrow \upuparrows\rangle$

A_{mov}

$A1.\ \langle e \to J; e \to e \rangle$
$A2.\ \langle J \leftrightarrow M_{LA}; e \to e \rangle;$ $A10.\ \langle J \leftrightarrow M_{RA}; e \to e \rangle;$
$A3.\ \langle M_{LA} \to m_{LA}; e \to \downarrow\downarrow \rangle$ $A11.\ \langle M_{RA} \to m_{RA}; e \to e \rangle$
$A4.\ \langle m_{LA} \leftrightarrow A; \downarrow\downarrow \leftrightarrow J \rangle$ $A12.\ \langle e \leftrightarrow J; m_{RA} \leftrightarrow b' \rangle$ b' is b if $b \in E$ or b' is ${}_A\bar{b}_d$
$A5.\ \langle A \to A; J \to J_0 \rangle$ $A13.\ \langle J \to J^0; b' \to O \rangle$ O is b if $b \in E$ or O is B
$A6.\ \langle A \leftrightarrow a; J_0 \leftrightarrow m_{LA} \rangle$ $A14.\ \langle O \leftrightarrow A; J_0 \leftrightarrow m_{RA} \rangle$
$A7.\ \langle m_{LA} \to m^0_{LA}; a \to a \rangle$ $A15.\ \langle A \to A; m_{RA} \to \downarrow\downarrow \rangle$
$A8.\ \langle a \leftrightarrow J_0; m^0_{LA} \to e \rangle$ $A16.\ \langle A \leftrightarrow e; \downarrow\downarrow \leftrightarrow J_0 \rangle$
$A9.\ \langle J^0 \to J; e \to e \rangle$ $A17.\ \langle J_0 \to J; e \to e \rangle$

If the rule is death of the form $(a; A_i; b) \to (a; \varepsilon; b)$, for $a; b \in E \cup N \cup \{\#\}$ the programs are following:

A_1

$61.\ \langle\ {}_aA_b \to e; \downarrow\downarrow \leftrightarrow \uparrow\uparrow \rangle$

In the last phase agent A_1 detects that the whole string is rewritten and it is time to non-deterministically choose if computation will be halted or will continue by simulating of following derivation step.

A_1

$62.\ \langle \uparrow\uparrow \leftrightarrow \downarrow\downarrow; e \leftrightarrow \# \rangle$
$63.\ \langle \downarrow\downarrow \to U; \# \leftrightarrow \uparrow\uparrow \rangle$
$64.\ \langle U \to V; \uparrow\uparrow \to \downarrow \rangle$
$65.\ \langle U \to F; \uparrow\uparrow \to F \rangle$
$66.\ \langle V \leftrightarrow \#'; \downarrow \leftrightarrow e \rangle$
$67.\ \langle \#' \leftrightarrow V; e \to \uparrow \rangle$
$68.\ \langle V \to e; \uparrow \to \uparrow \rangle$

The computation of the APCol system can halt only when in corresponding state of the PM-colony there is no applicable rule or in the case that program 56. is executed.

4 Boolean Circuits and APCol Systems

In this part, we show how APCol systems can simulate the functioning of logic gates. The input is always inserted into the string. Because there is no inner structure in the APCol systems, we use direct addressing. It means that every input symbol obtains index determining which agent will consume it. The result will be find on the string at the end of computation.

Theorem 2. *The functioning of logic gates NOT, OR and AND can be simulated by APCol systems with only one agent.*

Proof. The first we show how APCol system for logic gate NOT is constructed. Let NOT gate has an input with index i. We can simulate NOT gate with only one agent with following restricted programs:

1. $\langle e \leftrightarrow 0_i; e \rightarrow 1_{out} \rangle$ 3. $\langle e \leftrightarrow 1_i; e \rightarrow 0_{out} \rangle$
2. $\langle 0_i \rightarrow e; 1_{out} \leftrightarrow e \rangle$ 4. $\langle 1_i \rightarrow e; 0_{out} \leftrightarrow e \rangle$

Formally, we define APCol system $\Pi_{NOT} = (O, e, A_{NOT})$,
where $O = \{e, 1_i, 0_i, 1_{out}, 0_{out}\}, A_{NOT} = (ee, P_{AND})$, the set of programs is described above and initial string is formed from only one symbol 1_i or 0_i. The result is obtained after computation halts and it is placed in the string.

Simulation of the AND gate is done by APCol system with only one agent and following programs:

5. $\langle e \leftrightarrow 0_i; e \leftrightarrow 0_i \rangle$ 6. $\langle e \leftrightarrow 1_i; e \leftrightarrow 0_i \rangle$ 7. $\langle e \leftrightarrow 1_i; e \leftrightarrow 0_i \rangle$
8. $\langle e \leftrightarrow 1_i; e \leftrightarrow 1_i \rangle$ 9. $\langle 0_i \rightarrow 0_{out}; 0_i \rightarrow e \rangle$ 10. $\langle 1_i \rightarrow 0_{out}; 0_i \rightarrow e \rangle$
11. $\langle 1_i \rightarrow 1_{out}; 1_i \rightarrow e \rangle$

These programs can work only for boolean circuit with only one gate, because we expect that symbols of input are neighbouring. If we change the programs in such a way that they contains at most one communication rule. In this case they do not use context (one agent changes only one symbol on the string in one step).

12. $\langle e \leftrightarrow 0_i; e \rightarrow e \rangle$ 13. $\langle e \leftrightarrow 1_i; e \rightarrow e \rangle$ 14. $\langle e \leftrightarrow 0_i; 0_i \rightarrow 0 \rangle$
15. $\langle e \leftrightarrow 1_i; 1_i \rightarrow 1 \rangle$ 16. $\langle e \leftrightarrow 1_i; 0_i \rightarrow 0 \rangle$ 17. $\langle e \leftrightarrow 0_i; 1_i \rightarrow 1 \rangle$
18. $\langle 0_i \rightarrow 0_{out}; 0 \rightarrow e \rangle$ 19. $\langle 1_i \rightarrow 1_{out}; 1 \rightarrow e \rangle$ 20. $\langle 1_i \rightarrow 0_{out}; 0 \rightarrow e \rangle$
21. $\langle 0_i \rightarrow 0_{out}; 1 \rightarrow e \rangle$ 22. $\langle 0_{out} \leftrightarrow e; e \rightarrow e \rangle$ 23. $\langle 1_{out} \leftrightarrow e; e \rightarrow e \rangle$

Formally, we define APCol system $\Pi_{AND} = (O, e, A_{AND})$,
where $O = \{e, 1_i, 0_i, 1_{out}, 0_{out}\}, A_{AND} = (ee, P_{AND})$, the set of programs is formed from programs 12.-23. and initial string contains only one symbol 1_i or 0_i. The result is obtained after computation halts and it is placed in the string.

Simulation of the OR gate is done with programs very similar to programs of agent simulating AND gate. It is done by APCol system with only one agent and following programs:

24. $\langle e \leftrightarrow 0_i; e \rightarrow e \rangle$ 25. $\langle e \leftrightarrow 1_i; e \rightarrow e \rangle$ 26. $\langle e \leftrightarrow 0_i; 0_i \rightarrow 0 \rangle$
27. $\langle e \leftrightarrow 1_i; 1_i \rightarrow 1 \rangle$ 28. $\langle e \leftrightarrow 1_i; 0_i \rightarrow 0 \rangle$ 29. $\langle e \leftrightarrow 0_i; 1_i \rightarrow 1 \rangle$
30. $\langle 0_i \rightarrow 0_{out}; 0 \rightarrow e \rangle$ 31. $\langle 1_i \rightarrow 1_{out}; 1 \rightarrow e \rangle$ 32. $\langle 1_i \rightarrow 1_{out}; 0 \rightarrow e \rangle$
33. $\langle 0_i \rightarrow 1_{out}; 1 \rightarrow e \rangle$ 34. $\langle 0_{out} \leftrightarrow e; e \rightarrow e \rangle$ 35. $\langle 1_{out} \leftrightarrow e; e \rightarrow e \rangle$

Formally, we define APCol system $\Pi_{OR} = (O, e, A_{OR})$,
where $O = \{e, 1_i, 0_i, 1_{out}, 0_{out}\}, A_{OR} = (ee, P_{AND})$, the set of programs is formed from programs 24.-35. and initial string contains only one symbol 1_i or 0_i. The result is also obtained after computation halts and it is placed in the string.

We can proceed to formulate a Boolean circuit simulation theorem:

Theorem 3. *For every Boolean circuit there exists an APCol system that can simulate its functioning and for input string formed from input signals of Boolean circuit APCol system generates correct output.*

Let $\alpha = (V, E, \lambda)$ be a boolean circuit. We associate with each gate a label from a set $\{1, \ldots, |V|\}$. Let $(x_1, \ldots, x_p)$ be the vector of input signals for the boolean circuit. The correspoding initial string of APCol system is formed from objects $x_{0j}; 1 \leq j \leq p$. For simulation of the whole boolean circuit, we cannot only put the corresponding agents together. The output of one agent can be the input for

more agents. To obtain more input symbols we modify input objects consumed by agents (we use X'_i instead of X_i), output objects are in a form X_i – output from the agent i. We also add one more agent generating inputs for those agents which are connected with output of i-th agent. We deal with three cases: i-th agent is connected with one agent, with two agents or with m agents, $m > 2$. Let y be the index of agent with output and $zi$for $1 \leq i \leq m$ are indices of agents with input connected to agent y or corresponding to the initial input; $X \in \{0, 1\}$. We construct programs for agent $A_{|V|+1} = (ee, P_{|V|+1})$ generating inputs.

Programs for three variants of generated number of inputs

One input	Two inputs
36. $\langle e \leftrightarrow X_y; e \rightarrow X'_{z1} \rangle$	38. $\langle e \leftrightarrow X_y; e \rightarrow X'_{z1} \rangle$
37. $\langle X'_{z1} \leftrightarrow e; X_y \rightarrow e \rangle$	39. $\langle X'_{z1} \leftrightarrow e; X_y \rightarrow X'_{z2} \rangle$
	40. $\langle X'_{z2} \leftrightarrow e; e \rightarrow e \rangle$

m inputs; $m > 2$	
41. $\langle e \leftrightarrow X_y; e \rightarrow X'_{z1} \rangle$	
42. $\langle X'_{z1} \leftrightarrow e; X_y \rightarrow X_{2y} \rangle$	
43. $\langle e \rightarrow X'_{z2}; X_{2y} \rightarrow X_{2y} \rangle$	
44. $\langle X'_{z2} \leftrightarrow e; X_{2y} \rightarrow X_{3y} \rangle$	
45. $\langle e \rightarrow X'_{zk}; X_{ky} \rightarrow X_{ky} \rangle$	for $2 < k \leq m$
46. $\langle X'_{zk} \leftrightarrow e; X_{ky} \rightarrow X_{(k+1)y} \rangle$	for $2 < k \leq m - 1$
47. $\langle X'_{zm} \leftrightarrow e; X_{my} \rightarrow e \rangle$	

The agents associated to logic gates can consume only input symbols of a type X'_i, where X is 0 or 1 and i is label of gate. At the beginning of computation there are only symbols of type X_i present in the input string of APCol system. The only agent with applicable program is agent $A_{|V|+1}$ – program 36., 38. or 41. In the next step agent $A_{|V|+1}$ put the first input symbol ($X'_z k$), where zk is label of corresponding gate. From this step agents associated with gates started to do their work – rewriting their own inputs into outputs – and agent $A_{|V|+1}$ is rewriting their output to inputs. If the boolean circuit is defined correctly, at the end of computation there are only symbols X'_{out} in the string corresponding to the output of boolean circuit.

5 Conclusion

In this paper we continue with our research devoted to relationship of APCol systems which are devices derived from P colonies and P systems in general with classical formal models used not only in theoretical computer science. For this paper we have chosen PM-colonies and boolean circuits. We showed that APcol systems with two agents can simulate PM-colonies with strict conflict solving. In the second part we showed that even APCol systems are without inner structure they can substitute boolean circuits.

Acknowledgements.

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602, and by the Silesian University in Opava under the Student Funding Scheme, project SGS/13/2016.

References

1. Ceterchi, R., Sburlan, D.: Simulating Boolean Circuits with P Systems, pp. 104–122. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), http://dx.doi.org/10.1007/ 978-3-540-24619-0_8
2. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: A class of restricted p colonies with string environment. Natural Computing 15(4), 541–549 (2016), http://dx.doi.org/10.1007/ s11047-016-9564-3
3. Cienciala, L., Ciencialová, L., Kelemenová, A.: On the number of agents in P colonies. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 193–208. Springer (2007)
4. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: Towards on P colonies processing strings. In: Proc. BWMC 2014, Sevilla, 2014. pp. 102–118. Fénix Editora, Sevilla, Spain (2014)
5. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E., Vaszil, G.: Pcol automata: Recognizing strings with P colonies. In: Proc. BWMC 2010, Sevilla, 2010. pp. 65–76. Fénix Editora, Sevilla, Spain (2010)
6. Cienciala, L., Ciencialová, L., Kelemenová, A.: Homogeneous P colonies. Computing and Informatics 27(3+), 481–496 (2008)
7. Ciencialová, L., Cienciala, L.: Variation on the theme: P colonies. In: Proceedings of the International Workshop on Formal Models Prerov, the Czech Republic, April 25-27, 2006. pp. 27–34. CEUR Workshop Proceedings, Ostrava (2006)
8. Ciencialová, L., Csuhaj-Varjú, E., Kelemenová, A., Vaszil, G.: Variants of P colonies with very simple cell structure. International Journal of Computers, Communication and Control 4(3), 224–233 (2009)
9. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A.: Computing with cells in environment: P colonies. Multiple-Valued Logic and Soft Computing 12(3-4), 201–215 (2006)
10. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: P colonies with a bounded number of cells and programs. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4361, pp. 352–366. Springer (2006)
11. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory, EATCS Monographs in Theoretical Computer Science, vol. 18. Springer-Verlag Berlin (1989)
12. Freund, R., Oswald, M.: P colonies working in the maximally parallel and in the sequential mode. In: in G. Ciobanu, Gh. Păun, Pre-Proc. of First International Workshop on Theory and Application of P Systems, Timisoara, Romania, September 26-27. pp. 49–56 (2005)
13. Freund, R., Oswald, M.: P colonies and prescribed teams. Int. J. Comput. Math. 83(7), 569–592 (2006)

14. Gheorghe, M., Konur, S., Ipate, F.: Kernel P Systems and Stochastic P Systems for Modelling and Formal Verification of Genetic Logic Gates, pp. 661–675. Springer International Publishing, Cham (2017), http://dx.doi.org/ 10.1007/ 978-3-319-33924-5_25
15. Greibach, S.: Remarks on blind and partially blind one-way multicounter machines. Theoretical Computer Science 7(3), 311 – 324 (1978)
16. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation (3rd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006)
17. Kelemen, J., Kelemenová, A.: A grammar-theoretic treatment of multiagent systems. Cybern. Syst. 23(6), 621–633 (November 1992)
18. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX). pp. 82–86. Boston, Massachusetts, USA (September 12-15 2004)
19. Kelemenová, A.: P colonies. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 584–593. Oxford University Press (2010)
20. Martín-Vide, C., Păun, G.: Pm-colonies. Computers and Artificial Intelligence 17(6) (1998)
21. Meduna, A., Zemek, P.: Jumping finite automata. Int. J. Found. Comput. Sci. 23(7), 1555–1578 (2012)
22. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1967)
23. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
24. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000), http://www.sciencedirect.com/ science/ article/ pii/S0022000099916938
25. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. 1: Word, Language, Grammar. Springer-Verlag New York, Inc., New York, NY, USA (1997)
26. Vavrecková, Š., Cienciala, L., Ciencialová, L.: About Models Derived from Colonies, pp. 369–386. Springer International Publishing, Cham (2015), http://dx.doi.org/ 10.1007/ 978-3-319-28475-0_25

Subroutines in P Systems and Closure Properties of Their Complexity Classes*

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

Dipartimento di Informatica, Sistemistica e Comunicazione
Universit degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,luca.manzoni,mauri,porreca,zandron}@disco.unimib.it

Summary. The literature on membrane computing describes several variants of P systems whose complexity classes $\boldsymbol{C}$ are "closed under exponentiation", that is, they satisfy the inclusion $\mathbf{P}^{\boldsymbol{C}} \subseteq \boldsymbol{C}$, where $\mathbf{P}^{\boldsymbol{C}}$ is the class of problems solved by polynomial-time Turing machines with oracles for problems in $\boldsymbol{C}$. This closure automatically implies closure under many other operations, such as regular operations (union, concatenation, Kleene star), intersection, complement, and polynomial-time mappings, which are inherited from $\mathbf{P}$. Such results are typically proved by showing how elements of a family of P systems $\boldsymbol{\Pi}$ can be embedded into P systems simulating Turing machines, which exploit the elements of $\boldsymbol{\Pi}$ as subroutines. Here we focus on the latter construction, abstracting from the technical details which depend on the specific variant of P system, in order to describe a general strategy for proving closure under exponentiation.

1 Introduction

Complexity classes of the form $\mathbf{P}^{\boldsymbol{C}}$, characterised by polynomial-time Turing machines with oracles for languages in $\boldsymbol{C}$ [19], automatically inherit from $\mathbf{P}$ many closure properties. For instance, the determinism of Turing machines characterising $\mathbf{P}$ implies closure under complement, simply by switching the accepting and rejecting states of the machine. Under certain assumptions on $\boldsymbol{C}$, further closure properties are satisfied.

Let us say that a "reasonable" k-ary operation on languages is a function $f\colon \left(2^{\Sigma^\star}\right)^k \to 2^{\Sigma^\star}$ such that $f(L_1, \ldots, L_k) \in \mathbf{P}^{L_1,\ldots,L_k}$, that is, it can be computed efficiently with oracles for the k languages. A class $\boldsymbol{C}$ is closed under reasonable operations if $f(L_1, \ldots, L_k) \in \boldsymbol{C}$ for each reasonable operation f and for each $L_1, \ldots, L_k \in \boldsymbol{C}$. The reasonable operations include all usual set theoretic

* This work was partially supported by Fondo d'Ateneo 2016 of Universit degli Studi di Milano-Bicocca, project 2016-ATE-0492 "Sistemi a membrane: classi di complessit spaziale e temporale".

ones (complement, union, intersection and all derived operations) and common language theoretic operations, such as the regular ones (concatenation, union, and Kleene star) [9].

For instance, a class $\boldsymbol{C}$ of the form $\mathbf{P}^{\boldsymbol{D}}$ is closed under reasonable operations whenever $\boldsymbol{D}$ is an *upward directed set*, that is, for each $L_1, L_2 \in \boldsymbol{D}$ there exists $L \in \boldsymbol{D}$ such that $L_1 \leq L$ and $L_2 \leq L$, where $\leq$ denotes polynomial-time reducibility; an oracle for L can thus answer queries for both L_1 and L_2 after a polynomial-time reduction. Any class $\boldsymbol{D}$ with a complete problem L is directed, since all languages in $\boldsymbol{D}$ can be reduced to L. This shows that classes such as $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{coNP}}$ and $\mathbf{P}^{\mathbf{PP}}$ (which coincides with $\mathbf{P}^{\#\mathbf{P}}$) are closed under reasonable operations. Furthermore, it clearly suffices to find a subset $\boldsymbol{E} \subseteq \boldsymbol{D}$ with $\mathbf{P}^{\boldsymbol{E}} = \mathbf{P}^{\boldsymbol{D}}$ and prove $\mathbf{P}^{\boldsymbol{E}}$ closed under reasonable operations to obtain the same result for $\mathbf{P}^{\boldsymbol{D}}$. This is the case for $\boldsymbol{D} = \mathbf{NP} \cup \mathbf{coNP}$ (and $\boldsymbol{E} = \mathbf{NP}$), a class that frequently appears in the membrane computing literature [8, 21, 23, 31, 30] and is not known to be itself directed. In fact this would imply $\mathbf{NP} = \mathbf{coNP}$, since there would be a language $L \in \mathbf{NP} \cup \mathbf{coNP}$ such that $L_1 \leq L$ for an **NP**-complete language L_1 and $L_2 \leq L$ for a **coNP**-complete language L_2. Other classes $\boldsymbol{C}$ trivially closed under reasonable operations are those satisfying $\mathbf{P}^{\boldsymbol{C}} = \boldsymbol{C}$, such as **PH**, the polynomial hierarchy [29], and **CH**, the counting hierarchy [33].

Several variants of P systems have been proved able to simulate polynomial-time Turing machines with oracles, exploiting a family $\boldsymbol{\Pi}$ of P systems deciding a language L as "subroutines", by embedding them into the membrane structure of larger P systems providing the input and processing the output of the elements of $\boldsymbol{\Pi}$ [24, 12, 13, 16, 14]. This implies the closure under exponentiation of the corresponding complexity classes, in symbols $\mathbf{P}^{\mathbf{PMC}_{\mathcal{D}}} \subseteq \mathbf{PMC}_{\mathcal{D}}$, for some kind of P system $\mathcal{D}$. In this paper we describe this subroutine construction in a manner as independent from the specific variant of P systems as possible. In particular, we consider the cases of cell-like P systems (Section 2), tissue P systems (Section 2.1), and introduce a new construction for monodirectional cell-like P systems (Section 3).

Many computational complexity results in membrane computing have the form $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{D}}$, that is, some class $\mathcal{D}$ of P systems (e.g., active membranes without charges and dissolution using minimal cooperative rules [30, Corollary 6.6]) can solve in polynomial time all **NP** and **coNP** problems. We argue that this is unlikely to ever be an exact characterisation, since the features that allow us to solve **NP**-complete problems efficiently are the same that allow the subroutine construction, and this would imply $\mathbf{NP} = \mathbf{coNP}$.

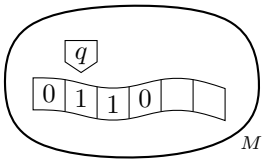
2 Subroutines in Cell-like P Systems

Several Turing machine simulations by means of polynomial-time uniform families of P systems have been proposed in the literature; some of these apply to unrestricted Turing machines [26, 2], while others are limited to machines working in logarithmic space [25], polynomial time [7, 6], polynomial

space [32, 24, 17, 10, 12, 13, 15, 16, 14], or exponential space [1]. Most of these solutions [32, 24, 1, 2, 7, 17, 10, 12, 13, 15, 6, 16, 14] are able to simulate Turing machines working in polynomial time with a polynomial slowdown.

The current configuration of the simulated Turing machine can be encoded in several equivalent ways by the simulating P system. A simple, common encoding [12, 13, 15, 16, 14] for a configuration of a polynomial-space Turing machine in state q, having the tape head in position i, and the tape containing the string $x = x_1 \cdots x_m$ is given by the multiset $q_i x_{1,1} \cdots x_{m,m}$, where the object q_i encodes state and head position, and the symbol x_j contained in tape cell j is encoded by the object $x_{j,j}$ of the multiset (i.e., it is subscripted with its position on the tape). The simplest mechanism to simulate a step of the Turing machine is using cooperative rewriting rules; suppose that $\delta(q, a) = (r, b, +1)$ describes the transition of the machine reading symbol a in state q to state r, symbol b and movement to the right. This can be trivially simulated by the cooperative rewriting rule $[q_i\ a_i \to r_{i+1}\ b_i]_M$, which is repeated for each cell position i up to the maximum length of the tape. Notice that *minimal cooperation*, with only two objects on the left-hand side of the rules, suffices for this purpose [31, 30]. All published solutions known to the authors [12, 13, 15, 16, 14] use alternative mechanisms (such as membrane charges, antimatter annihilation, antiport communication) to perform essentially the same operation.

Irrespective of the actual encoding of the configuration of the Turing machine and the mechanism employed to simulate a computation step, we will use the following symbol



to denote a membrane structure with root M inside which the simulation is carried over.

Let us now consider the case of a Turing machine M with an oracle for language $L \subseteq \Sigma^\star$. Suppose that M writes on its tape the query string $x \in \Sigma^\star$ and enters its query state. Suppose that language L is decided by a family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of recogniser P systems [22, 18]. By embedding the (empty) membrane structure of the P system Π_x inside the membrane structure of the P system simulating M (Fig. 1) and initialising the configuration of Π_x when the simulation reaches the query state, we can simulate the oracle and read the result of the query as the output of Π_x [24]. It is possible to send-in the initial multisets contained in Π_x, which typically requires some synchronisation using timers, so that all initial objects appear simultaneously (Fig. 2).

Since the query string x is, in general, unknown before the beginning of the simulation, several P systems must be embedded in order to be able to process query strings of different length. Suppose that language L is decided by a uniform

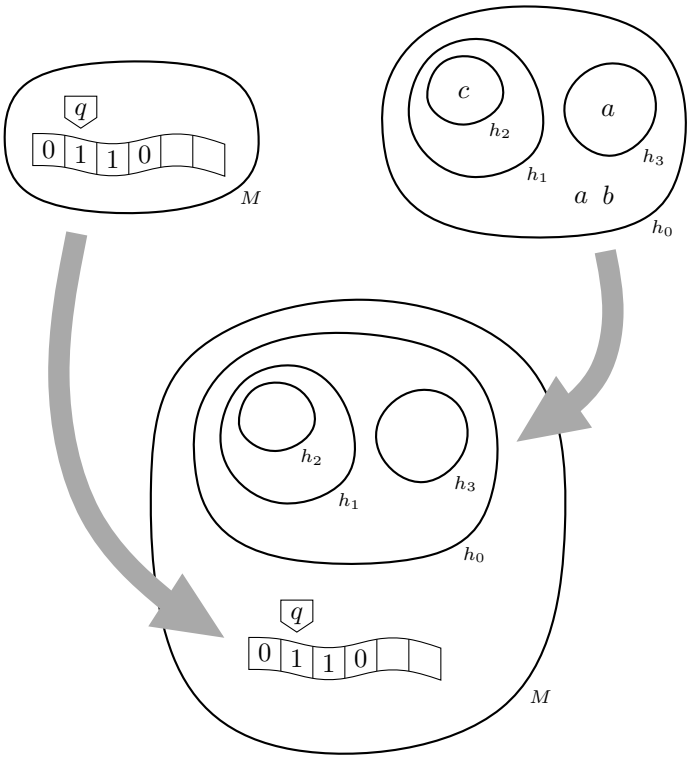


Fig. 1. The (empty) membrane structure of a P system is embedded into another P system simulating a Turing machine M.

family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of P systems, where input strings of the same length n are associated to P systems sharing the same membrane structure and rules $\Pi(n)$, which only differ with respect to the initial multisets they contain. Then, multiple empty membrane structures $\Pi(0), \Pi(1), \ldots, \Pi(m)$ can be embedded, up to the maximum possible query string length (an upper bound is given, for instance, by the length of the tape of M), and the correct one is selected at runtime by the P system simulating M (Fig. 3).

When the simulated Turing machine performs multiple queries with query strings of the same length during its computation, it is possible to embed multiple copies of each P system $\Pi(n)$ [24]. Each of these copies can then be used for a single query (Fig. 4). A possible alternative is to reset the configuration of P system $\Pi(n)$ after the query simulation has been performed [12].

2.1 Subroutines in Tissue P Systems

The oracle query construction for cell-like P systems embeds elements of a family of P systems into a membrane where a Turing machine is simulated. This construc-

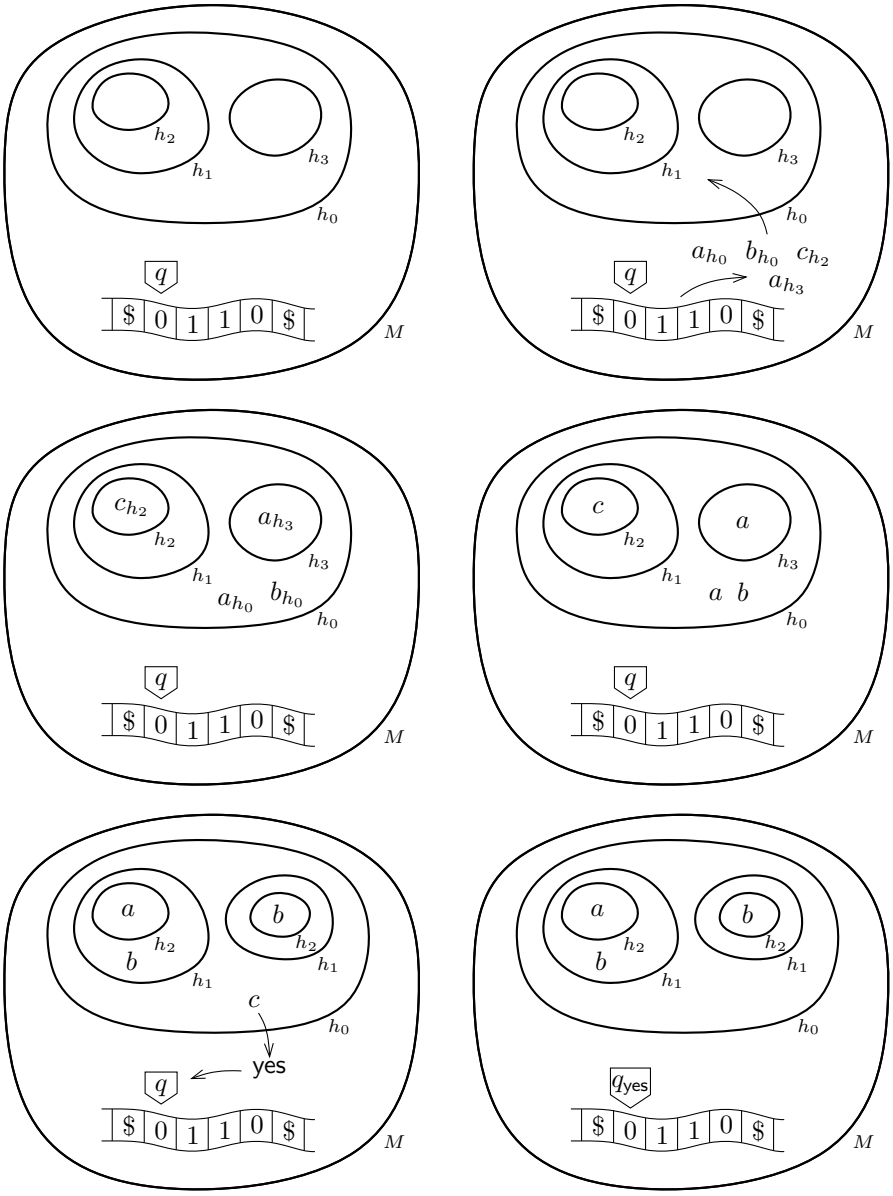


Fig. 2. The query string x on the tape of the simulated Turing machine is encoded as the initial configuration of the embedded P system Π_x. Each object is subscripted by the label of its target membrane, which it reaches by using send-in rules. When all objects have reached their destination, they simultaneously lose the subscripts, and the computation of Π_x begins. The output of Π_x is then read and incorporated into the state of the simulated Turing machine, as in the configuration following the query.

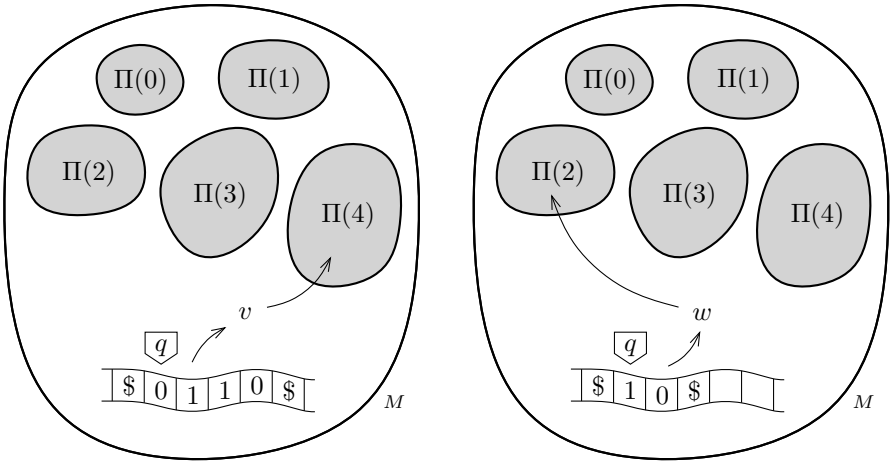


Fig. 3. Query strings of different lengths can be processed by distinct embedded P systems, selected when the oracle query is simulated.

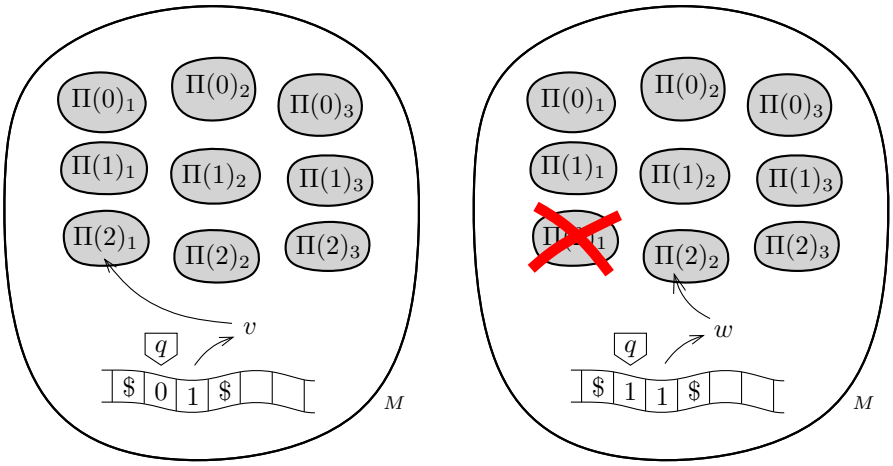


Fig. 4. Multiple query strings of the same length can be processed by replicating the P systems simulating the oracle and using each of them only once.

tion can be adapted to tissue P systems by having the Turing machine simulation take place in a cell, and placing the elements of the family of tissue P systems deciding the oracle language on the side [16]. The communication needed in order to simulate the oracle queries in this variant is not hierarchical, but between adjacent cells (Fig. 5).

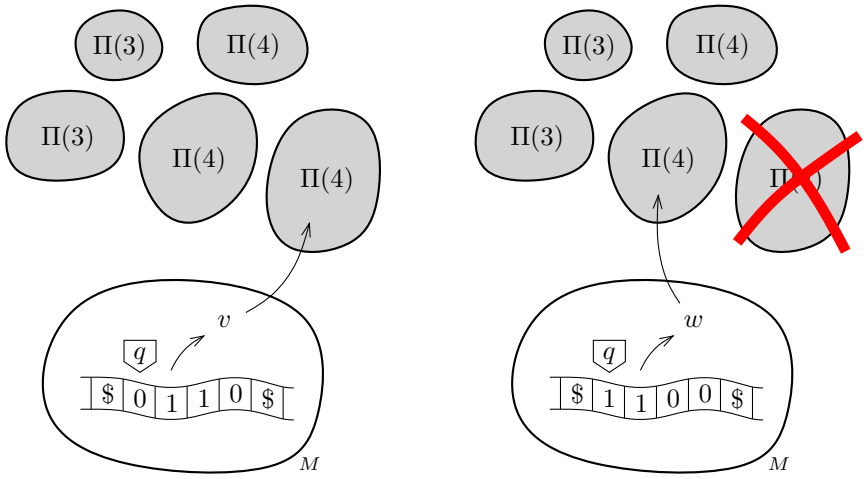


Fig. 5. Simulating oracle queries in tissue P systems.

3 Subroutines in Monodirectional Cell-like P Systems

In monodirectional cell-like P systems [13, 14], where send-in rules are disallowed, the construction of Section 2 does not apply. However, we can turn the construction of Fig. 1 inside out, by having the Turing machine simulation embedded *into the input membrane* of the P system Π simulating the oracle (Fig. 6). Instead of sending in the encoding of the query string, it is the encoding of the configuration of the Turing machine that is sent *out*, through the whole membrane structure of Π, where it waits for the oracle query result (Fig. 7) [13, 14]. This is needed because the P system simulating the oracle can only send its result outwards and, since the system is monodirectional, the only way to intercept it is to move out the entire simulation of the Turing machine.

Unlike the bidirectional case, the objects of the initial configuration of Π cannot be arranged during the query simulation, as that requires send-in if the membrane structure of Π is not linear or if the input membrane is not elementary. A solution is to have them already in their correct position in the initial configuration of the combined P system but with a timer subscript, which is deleted at a predefined time step t, when a query may take place. If the simulated Turing machine does not perform a query at time t, it can be adapted so that it makes a "dummy" query at that time, ignoring its result.

As in the bidirectional case, the oracle strings are usually only available at runtime, and thus multiple P systems simulating oracles must be arranged in advance, in order to accommodate any possible query string. Given the restriction on the direction of communication, a natural solution is to *nest* these auxiliary P systems, placing each one inside the input membrane of the next one. When a query takes place, the Turing machine configuration is first moved to the first P system suitable for the query string, where the multiset encoding it is left;

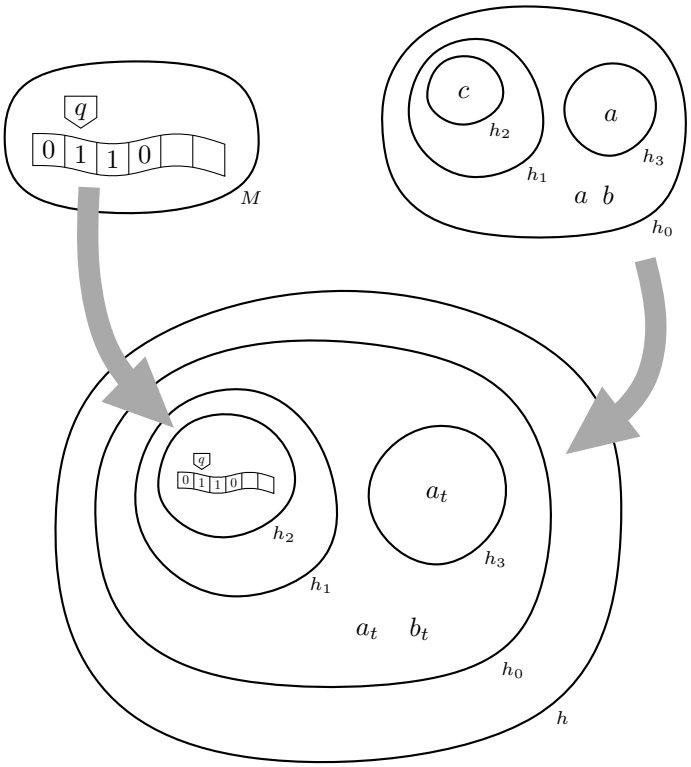


Fig. 6. In the monodirectional case, the P system simulating the Turing machine is embedded into the input membrane of the P system Π simulating the oracle; the other objects in the initial configuration of Π have an associated timer for synchronisation purposes.

then, the Turing machine configuration is moved outside the whole set of auxiliary P systems, where it waits for the result of the query (Fig. 7).

If multiple queries are carried out, it suffices to repeatedly nest the array of auxiliary P systems, always using the input membranes as junction points, and repeating the query procedure as many times as necessary (Fig. 8) [13, 14].

4 Discussion and Open Problems

Many complexity theory results for P systems have the form $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{D}}$ for some class $\mathcal{D}$ of P systems, and by closure under complementation this implies $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{D}}$ [22]. Solving **NP**-complete problems usually requires some form of "context-sensitivity", such as cooperative evolution rules (even minimal cooperation), membranes charges, membrane dissolution, or antimatter annihilation [31, 30, 20, 4, 5]. However, these same features typically suffice to carry

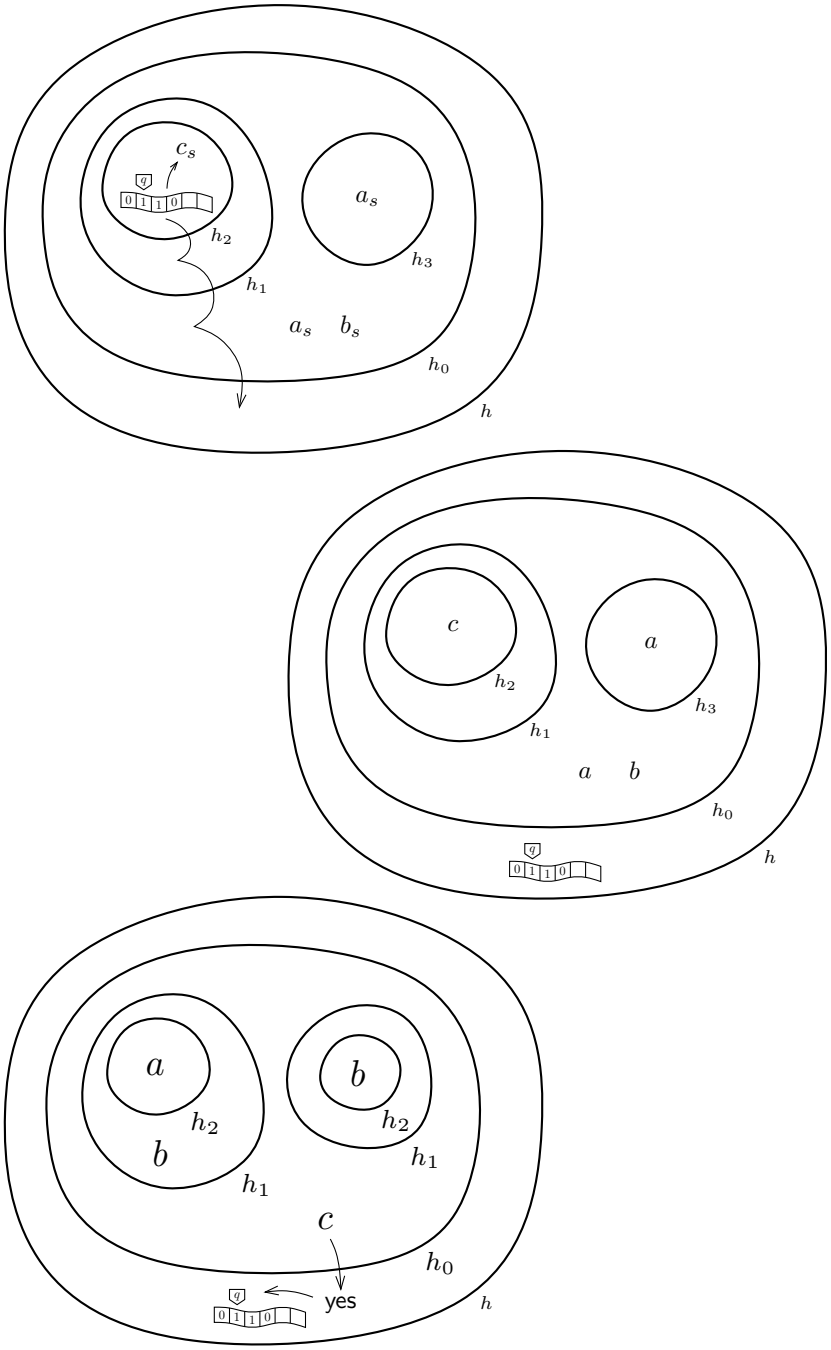


Fig. 7. Query simulation procedure for monodirectional cell-like P systems.

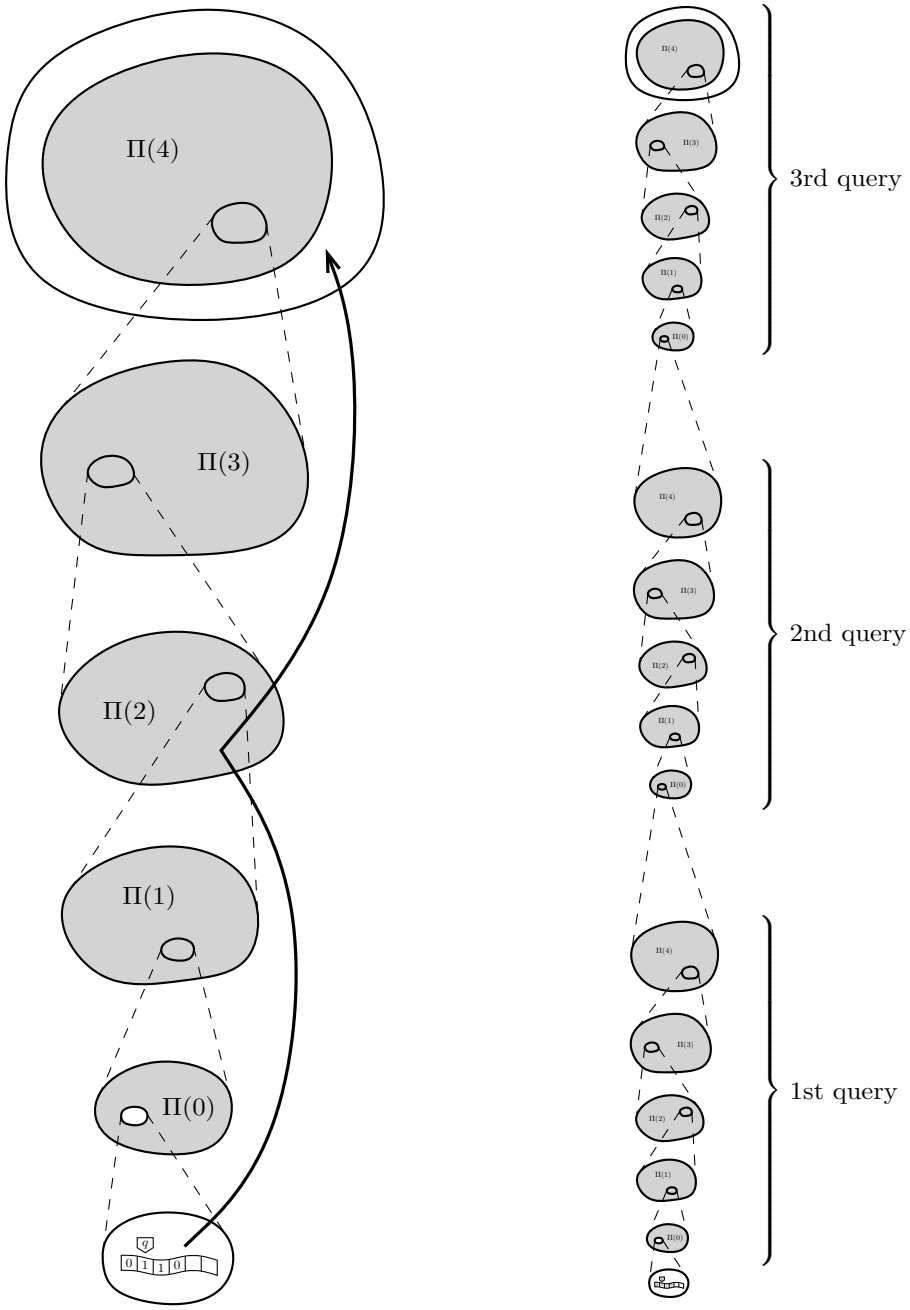


Fig. 8. Repeated nesting of monodirectional cell-like P systems to perform queries of a priori unknown length (on the left) and for multiple queries of a priori unknown length (on the right).

out the oracle simulation constructions described in Section 2 or 3, which imply that $\mathbf{PMC}_{\mathcal{D}}$ has the form $\mathbf{P}^{\boldsymbol{C}}$ for some class $\boldsymbol{C}$. This means that it is quite unlikely that $\mathbf{NP} \cup \mathbf{coNP}$ is an exact characterisation of $\mathbf{PMC}_{\mathcal{D}}$.

Indeed, if $\mathbf{NP} \cup \mathbf{coNP}$ were of the form $\mathbf{P}^{\boldsymbol{C}}$, then $\mathbf{P}^{\mathbf{NP} \cup \mathbf{coNP}} = \mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}}$. But $\mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}} = \mathbf{P}^{\boldsymbol{C}}$: if $L_1 \in \mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}}$, there exists a Turing machine M_1 with oracle for $L_2 \in \mathbf{P}^{\boldsymbol{C}}$ such that $L(M_1) = L_1$, and a Turing machine M_2 with an oracle for $L_3 \in \boldsymbol{C}$ such that $L_2 = L(M_2)$. Let M be a Turing machine with oracle for L_3. This machine simulates M_1 until it enters its query state, then it simulates M_2 until it enters its own query state; since M and M_2 have the same oracle, the queries of M_2 can be answered directly. Then $L(M) = L(M_1) = L_1$, and we can conclude that $\mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}} = \mathbf{P}^{\boldsymbol{C}}$. But then $\mathbf{P}^{\mathbf{NP} \cup \mathbf{coNP}} = \mathbf{P}^{\boldsymbol{C}} = \mathbf{NP} \cup \mathbf{coNP}$. Since $\mathbf{P}^{\mathbf{NP} \cup \mathbf{coNP}} = \mathbf{P}^{\mathbf{NP}}$, this class has complete problems, for instance the standard complete problem H of deciding if a Turing machine M with **NP** oracle accepts a string x within t steps (where t is given in unary notation). Then either $H \in \mathbf{NP}$ or $H \in \mathbf{coNP}$, and it is hard for both **NP** and **coNP**; this means that either there exists a **NP**-hard problem in **coNP**, or a **coNP**-hard problem in **NP**: in both cases, this implies $\mathbf{NP} = \mathbf{coNP}$.

Furthermore, it is often the case [24, 12, 14, 16] that the amount of context-sensitivity that allows us to simulate Turing machines with oracles also suffices, at least in the bidirectional case, to simulate not only oracles for decision problems, but their counting version, which is usually more powerful. For instance, several variants of P systems without non-elementary membrane division rules characterise the complexity class $\mathbf{P}^{\mathbf{PP}}$ (which coincides with $\mathbf{P}^{\#\mathbf{P}}$) [19, 11, 14, 16].

Finally, notice that the above remarks apply even to variants of P systems powerful enough to characterise **PSPACE** in polynomial time [27, 3, 28, 4], even if the algorithms developed in order to prove these results do not usually employ an oracle simulation construction. Indeed, the class **PSPACE** is closed under exponentiation: $\mathbf{P}^{\mathbf{PSPACE}} = \mathbf{PSPACE}$.

The property investigated in this paper, closure under exponentiation, that is, satisfying $\mathbf{P}^{\boldsymbol{C}} = \boldsymbol{C}$, seems to apply to a wide range of variants of P systems. A natural follow-up question is whether the more common *closure under oracles* (or *under subroutines*), where $\boldsymbol{C} = \boldsymbol{C}^{\boldsymbol{C}}$, does also apply for some of these variants. This also requires establishing a notion of "P system with oracle"; a first definition has been already given in [12], albeit for purely technical reasons.

An interesting open problem, although a presumably challenging one from a technical standpoint, is to formally characterise the membrane computing features (such as combination of rules or properties of the membrane structures) that enable the oracle simulation construction.

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: The computational power of exponential-space P systems with active membranes. In: Martínez-del-Amor, M.A., Păun, Gh., Pérez-Hurtado, I., Romero-Campero, F.J. (eds.) Pro-

ceedings of the Tenth Brainstorming Week on Membrane Computing, vol. I, pp. 35–60. Fénix Editora (2012)
2. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. Theoretical Computer Science 529, 69–81 (2014)
3. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundamenta Informaticae 58(2), 67–77 (2003)
4. Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution to QSAT using polarizationless active membranes. In: Durand-Lose, J., Margenstern, M. (eds.) Machines, Computations, and Universality, 5th International Conference, MCU 2007, Lecture Notes in Computer Science, vol. 4664, pp. 122–133. Springer (2007)
5. Díaz-Pernil, D., Alhazov, A., Freund, R., Gutiérrez-Naranjo, M.A., Leporati, A.: Recognizer P systems with antimatter. Romanian Journal of Information Science and Technology 18(3), 201–217 (2015)
6. Gazdag, Z., Kolonits, G.: Remarks on the computational power of some restricted variants of P systems with active membranes. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 209–232. Springer (2017)
7. Gazdag, Z., Kolonits, G., Gutiérrez-Naranjo, M.A.: Simulating Turing machines with polarizationless P systems with active membranes. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014. Lecture Notes in Computer Science, vol. 8961, pp. 229–240 (2014)
8. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. Theoretical Computer Science 371(1–2), 54–61 (2007)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd Edition. Addison-Wesley (2006)
10. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Constant-space P systems with active membranes. Fundamenta Informaticae 134(1–2), 111–128 (2014)
11. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: Gheorghe, M., Rozenberg, G., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014, Lecture Notes in Computer Science, vol. 8961, pp. 284–299. Springer (2014)
12. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. Fundamenta Informaticae 138(1–2), 97–111 (2015)
13. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing 15(4), 551–564 (2016)
14. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. Theoretical Computer Science (2017), in press
15. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Shallow nonconfluent P systems. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 307–316 (2017)

16. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E.P., Zandron, C.: Characterising the complexity of tissue P systems with fission rules. Journal of Computer and System Sciences (2017), in press
17. Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes. Journal of Automata, Languages and Combinatorics 19(1–4), 173–184 (2014)
18. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
19. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
20. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
21. Pérez-Jiménez, M.J.: The P versus NP problem from the membrane computing view. European Review 22(01), 18–33 (2014)
22. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
23. Pérez-Jiménez, M.J., Sosík, P.: An optimal frontier of the efficiency of tissue P systems with cell separation. Fundamenta Informaticae 138(1–2), 45–60 (2015)
24. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems simulating oracle computations. In: Gheorghe, M., Păun, Gh., Salomaa, A., Rozenberg, G., Verlan, S. (eds.) Membrane Computing, 12th International Conference, CMC 2011, Lecture Notes in Computer Science, vol. 7184, pp. 346–358. Springer (2012)
25. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing, 13th International Conference, CMC 2012, Lecture Notes in Computer Science, vol. 7762, pp. 342–357. Springer (2013)
26. Romero-Jiménez, A., Pérez-Jiméz, M.J.: Simulating Turing machines by P systems with external output. Fundamenta Informaticae 49(1–3), 273–287 (2002)
27. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
28. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
29. Stockmeyer, L.J.: The polynomial-time hierarchy. Theoretical Computer Science 3(1), 1–22 (1976)
30. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. Fundamenta Informaticae 153(1–2), 147–172 (2017)
31. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Reaching efficiency through collaboration in membrane systems: Dissolution, polarization and cooperation. Theoretical Computer Science (2017), in press
32. Valsecchi, A., Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 10th International Workshop, WMC 2009, Lecture Notes in Computer Science, vol. 6501, pp. 461–478. Springer (2010)

33. Wagner, K.W.: The complexity of combinatorial problems with succinct input representation. Acta Informatica 23(3), 325–356 (1986)

On Efficiency of P Systems with Symport/Antiport and Membrane Division

Luis F. Macías-Ramos[1], Bosheng Song[2],
Tao Song[2], Linqiang Pan[2], Mario J. Pérez-Jiménez[1]

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

[2] Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China
E-mail: boshengsong@163.com, songtao0608@hotmail.com,
lqpan@mail.hust.edu.cn

Summary. Classical membrane systems with symport/antiport rules observe the *conservation law*, in the sense that they compute by changing the places of objects with respect to the membranes, and not by changing the objects themselves. In these systems the environment plays an active role because the systems not only send objects to the environment, but also bring objects from the environment. In the initial configuration of a system, there is a special alphabet whose elements appear in an arbitrary large number of copies. The ability of these computing devices to have infinite copies of some objects has been widely exploited in the design of efficient solutions to computationally hard problems.

This paper deals with computational aspects of P systems with symport/antiport and membrane division rules where there is not an environment having the property mentioned above. Specifically, we establish the relationships between the polynomial complexity class associated with P systems with symport/antiport, membrane division rules, and with or without environment. As a consequence, we prove that the role of the environment is irrelevant in order to solve **NP**–complete problems in an efficient way.

Key words: Membrane Computing, P System with Symport/Antiport, Membrane Division, Computational Complexity.

1 Preliminaries

An *alphabet* Γ is a non–empty set whose elements are called *symbols*. An ordered finite sequence of symbols is a *string* or *word*. If u and v are strings over Γ, then so

is their *concatenation* uv, obtained by juxtaposition, that is, writing u and v one after the other. The number of symbols in a string u is the *length* of the string and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of all strings over an alphabet Γ is denoted by Γ^*. In algebraic terms, Γ^* is the free monoid generated by Γ under the operation of concatenation. Subsets of Γ^* are referred to as *languages* over Γ. The set of symbols occurring in a string $u \in \Gamma^*$ is denoted by $alph(u)$.

The *Parikh vector* associated with a string $u \in \Gamma^*$ with respect to the alphabet $\Sigma = \{a_1, \ldots, a_r\} \subseteq \Gamma$ is $\Psi_\Sigma(u) = (|u|_{a_1}, \ldots, |u|_{a_r})$, where $|u|_{a_i}$ denotes the number of occurrences of symbol a_i in string u. The application Ψ_Σ is called the *Parikh mapping* associated with Σ. Notice that, in this definition, the ordering of the symbols from Σ is relevant. If $\Sigma_1 = \{a_{i_1}, \ldots, a_{i_r}\} \subseteq \Gamma$, then we define $\Psi_{\Sigma_1}(u) = (|u|_{a_{i_1}}, \ldots, |u|_{a_{i_r}})$, for each $u \in \Gamma^*$.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \,|\, f(x) > 0\}$. A multiset is empty (resp. finite) if its support is the empty set (resp. a finite set). If $m = (A, f)$ is a finite multiset over A and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$. That is, superscripts indicate the multiplicity of each element, and if $f(x) = 0$ for $x \in A$, then element x is omitted. A finite multiset $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$ can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1, \ldots, a_k\}$. Nevertheless, all permutations of this string identify the same multiset m precisely. We denote by $\emptyset$ the empty multiset and we denote by $\mathcal{M}_f(\Gamma)$ the set of all finite multisets over Γ. Throughout this paper, we speak about "the finite multiset m" where m is a string, meaning "the finite multiset represented by the string m". If $m_1 = (A, f_1)$, $m_2 = (A, f_2)$ are multisets over A, then we define the union of m_1 and m_2 as $m_1 + m_2 = (A, g)$, where $g = f_1 + f_2$, that is, $g(a) = f_1(a) + f_2(a)$, for each $a \in A$.

For any sets A and B the *relative complement* $A \setminus B$ of B in A is defined as follows: $A \setminus B = \{x \in A \,|\, x \notin B\}$. For any set A we denote $|A|$ the cardinal (number of elements) of A, as usual.

In what follows, we assume the reader is already familiar with the basic notions and terminology of P systems. For details, see [7].

2 P Systems with Symport/Antiport Rules and Membrane Division

Cell division is an elegant process that enables organisms to grow and reproduce. Mitosis is a process of cell division which results in the production of two daughter cells from a single parent cell. Daughter cells are identical to one another and to the original parent cell. Through a sequence of steps, the replicated genetic material in a parent cell is equally distributed to two daughter cells. While there are some subtle differences, mitosis is remarkably similar across organisms.

Before a dividing cell enters mitosis, it undergoes a period of growth where the cell replicates its genetic material and organelles. Replication is one of the most important functions of a cell. DNA replication is a simple and precise process that creates two complete strands of DNA (one for each daughter cell) where only one existed before (from the parent cell).

Next, we introduce an abstraction of these operation in the framework of P systems with symport/antiport rules. In these models, the membranes are not polarized; the membranes obtained by division have the same labels as the original membrane, and if a membrane is divided, its interaction with other membranes or with the environment is locked during the division process. In some sense, this means that while a membrane is dividing it closes its communication channels.

Definition 1. *A P system with symport/antiport rules and membrane division of degree* $q \geq 1$ *is a tuple* $\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$*, where:*

1. Γ *is a finite* alphabet.
2. $\mathcal{E} \subseteq \Gamma$.
3. μ *is a membrane structure (a rooted tree) whose nodes are injectively labelled with* $1, 2 \ldots, q$.
4. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are multises over* Γ.
5. $\mathcal{R}_1, \cdots, \mathcal{R}_q$ *are finite set of rules of the following forms:*
   (a) Communication rules*:* $(u, out), (u, in), (u, out; v, in)$*, for* u, v *multisets over* Γ *and* $|u| + |v| > 0$*;*
   (b) Division rules*:* $[a]_i \rightarrow [b]_i[c]_i$*, where* $i \neq i_{out}$ *and* $a, b, c \in \Gamma$*;*
6. $i_{out} \in \{0, 1, \ldots, q\}$.

A P system with symport/antiport rules and membrane division

$$\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$$

of degree q can be viewed as a set of q membranes, labelled by $1, \ldots, q$, arranged in a hierarchical structure, such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ represent the finite multisets of objects initially placed in the q membranes of the system; (b) $\mathcal{E}$ is the set of objects initially located in the environment of the system, all of them available in an arbitrary number of copies; and (c) i_{out} represents a distinguished *region* which will encode the output of the system. We use the term *region* i $(0 \leq i \leq q)$ to refer to membrane i in the case $1 \leq i \leq q$ and to refer to the environment in the case $i = 0$.

A rule of the type (u, out) or (u, in) is called a *symport* rule. A rule of the type $(u, out; v, in)$, where $|u| + |v| > 0$, is called an *antiport* rule. A P system with symport rules (resp. with antiport rules) is a P system with only symport rules (resp. only antiport rules) as communication rules. The length of rule (u, out) or (u, in) (resp. $(u, out; v, in)$) is defined as $|u|$ (resp. $|u| + |v|$).

An *instantaneous description* or a *configuration* at an instant t of a P system with symport/antiport and membrane division is described by all multisets of objects over Γ associated with all the membranes present in the system, and the

multiset of objects over $\Gamma - \mathcal{E}$ associated with the environment at that moment. Recall that there are infinitely many copies of objects from $\mathcal{E}$ in the environment, and hence this set is not properly changed along the computation. The *initial configuration* is $(\mathcal{M}_1, \cdots, \mathcal{M}_q; \emptyset)$.

A rule $(u, out) \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}$ at an instant t if membrane i is in $\mathcal{C}$ and multiset u is contained in such membrane. When applying a rule $(u, out) \in \mathcal{R}_i$, the objects specified by u are sent out of membrane i into the region immediately outside (its father), this can be the environment in the case of the skin membrane.

A rule $(u, in) \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}$ at an instant t if membrane i is in $\mathcal{C}$ and multiset u is contained in the immediately upper region (its father), this is the environment in the case when the rule is associated with the skin membrane (the root of the tree μ). When applying a rule $(u, in) \in \mathcal{R}_i$, the multiset of objects u enters the region defined by the membrane i from the immediately upper region (its father), this is the environment in the case when the rule is associated with the skin membrane (the root of the tree μ).

A rule $(u, out; v, in) \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}$ at an instant t if membrane i is in $\mathcal{C}$ and multiset u is contained in such membrane, and multiset v is contained in the immediately upper region (its father). When applying a rule $(u, out; v, in) \in \mathcal{R}_i$, the objects specified by u are sent out of membrane i into the region immediately outside (its father), at the same time bringing the objects specified by v into membrane i.

A rule $[a]_i \to [b]_i[c]_i \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}$ at an instant t if the following holds: (a) membrane i is in $\mathcal{C}$; (b) object a is contained in such membrane; and (c) membrane i is neither the skin membrane nor the output membrane (if $i_{out} \in \{1, \ldots, q\}$). When applying a division rule $[a]_i \to [b]_i[c]_i$, under the influence of object a, the membrane with label i is divided into two membranes with the same label; in the first copy, object a is replaced by object b, in the second one, object a is replaced by object c; all the other objects residing in membrane i are replicated and copies of them are placed in the two new membranes. The output membrane i_{out} cannot be divided.

The rules of a P system with symport/antiport rules and membrane division are applied in a non-deterministic maximally parallel manner (at each step we apply a multiset of rules which is maximal, no further applicable rule can be added), with the following important remark: if a membrane divides, then the division rule is the only one which is applied for that membrane at that step; the objects inside that membrane do not evolve by means of communication rules. In other words, before division a membrane interrupts all its communication channels with the other membranes and with the environment. The new membranes resulting from division will interact with other membranes or with the environment only at the next step – providing that they do not divide once again. The label of a membrane precisely identifies the rules which can be applied to it.

Let us fix a P system with symport/antiport rules and membrane division Π. We say that configuration $\mathcal{C}_1$ yields configuration $\mathcal{C}_2$ in one *transition step*, denoted

by $\mathcal{C}_1 \Rightarrow_{\Pi} \mathcal{C}_2$, if we can pass from $\mathcal{C}_1$ to $\mathcal{C}_2$ by applying the rules from $\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_q$ following the previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that:

1. the first term of the sequence is the initial configuration of the system;
2. each non-initial configuration of the sequence is obtained from the previous configuration by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and
3. if the sequence is finite (called *halting computation*) then the last term of the sequence is a *halting configuration* (a configuration where no rule of the system is applicable to it).

All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region i_{out} in the halting configuration.

If $\mathcal{C} = \{\mathcal{C}_t\}_{t<r+1}$ of Π ($r \in \mathbb{N}$) is a halting computation, then the *length of* $\mathcal{C}$, denoted by $|\mathcal{C}|$, is r, that is, $|\mathcal{C}|$ is the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. We denote by $\mathcal{C}_t(i), 1 \leq i \leq q$, the multiset of objects over Γ contained in all membranes labelled by i (by applying division rules different membranes with the same label can be created) at configuration $\mathcal{C}_t$. We denote by $\mathcal{C}_t(0)$ the multiset of objects over $\Gamma \setminus \mathcal{E}$ contained in the environment at configuration $\mathcal{C}_t$. Finally, we denote by $\mathcal{C}_t^*$ the multiset $\mathcal{C}_t(0)+\mathcal{C}_t(1)+\cdots+\mathcal{C}_t(q)$.

Definition 2. *A P system with symport/antiport rules and membrane division* $\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$*, where* $\mathcal{E} = \emptyset$*, is called a P system with symport/antiport rules, membrane division and without environment.*

Usually, we omit the alphabet of the environment in the tuple describing such P system.

3 Recognizer P systems with symport/antiport rules

Let us recall that a *decision problem* is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X. Many abstract problems are not decision problems. For example, in *combinatorial optimization problems* some value must be optimized (minimized or maximized). In order to deal with such problems, they can be transformed into roughly equivalent decision problems by supplying a target/threshold value for the quantity to be optimized, and then asking whether this value can be attained.

There exists a correspondence between decision problems and formal languages. So that, the solvability of decision problems is defined through the recognition of the languages associated with them.

In order to study the computing efficiency of membrane systems, the notions from classical *computational complexity theory* are adapted for membrane computing, and a special class of cell-like P systems is introduced in [10]: *recognizer P systems* (called *accepting P systems* in a previous paper [9]).

Definition 3. *A* recognizer P system *with symport/antiport rules and membrane division of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$$

where:

- $(\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$ *is a P system with symport/antiport rules and membrane division of degree $q \geq 1$, as defined in the previous section;*
- *The working alphabet Γ has two distinguished objects* `yes` *and* `no`*, at least one copy of them present in some initial multisets $\mathcal{M}_1$, ..., $\mathcal{M}_q$, but none of them is present in $\mathcal{E}$;*
- *Σ is an (input) alphabet strictly contained in Γ such that $\mathcal{E} \cap \Sigma = \emptyset$;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are multisets over $\Gamma \setminus \Sigma$;*
- *$i_{in} \in \{1, \ldots, q\}$ is the input membrane;*
- *The output region i_{out} is the environment;*
- *All computations halt;*
- *If $\mathcal{C}$ is a computation of Π, then either object* `yes` *or object* `no` *(but not both) must have been released into the output region (the environment), and only at the last step of the computation.*

Definition 4. *A* recognizer P system *with symport/antiport rules, membrane division and without environment of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$$

where:

- $(\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$ *is a P system with symport/antiport rules and membrane division.*
- *The working alphabet Γ has two distinguished objects* `yes` *and* `no`*, at least one copy of them present in some initial multisets $\mathcal{M}_1$, ..., $\mathcal{M}_q$, but none of them is present in $\mathcal{E}$.*
- *$\mathcal{E} = \emptyset$.*
- *Σ is an (input) alphabet strictly contained in Γ such that $\mathcal{E} \cap \Sigma = \emptyset$.*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are multisets over $\Gamma \setminus \Sigma$.*
- *$i_{in} \in \{1, \ldots, q\}$ is the input membrane.*
- *$i_{out} \in \{1, \ldots, q\}$ is the output membrane.*
- *All computations halt.*
- *If $\mathcal{C}$ is a computation of Π, then either object* `yes` *or object* `no` *(but not both) must have been released into the output region, and only at the last step of the computation.*

For each multiset $m \in \mathcal{M}_f(\Sigma)$, the *computation of the system Π with input $m \in \mathcal{M}_f(\Sigma)$* starts from the configuration of the form $(\mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} + m, \ldots, \mathcal{M}_q; \emptyset)$, that is, the input multiset m has been added to the contents of the input membrane i_{in}, and we denote it by $\Pi + m$. Therefore, we have an initial configuration

associated with each input multiset m (over the input alphabet Σ) in this kind of systems.

Given a recognizer P system with symport/antiport rules (with or without environment) and a halting computation $\mathcal{C} = \{\mathcal{C}_t\}_{t<r+1}$ of Π ($r \in \mathbb{N}$), we define the result of $\mathcal{C}$ as follows:

$$Output(\mathcal{C}) = \begin{cases} \texttt{yes}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,i_{out}}) = (1,0) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{t,i_{out}}) = (0,0) \text{ for } t = 0,\ldots,r-1 \\ \texttt{no}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,i_{out}}) = (0,1) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{t,i_{out}}) = (0,0) \text{ for } t = 0,\ldots,r-1 \end{cases}$$

where Ψ is the Parikh mapping, and $M_{t,i_{out}}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the output region at the configuration $\mathcal{C}_t$, in particular, $M_{r,i_{out}}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the output region at the halting configuration $\mathcal{C}_r$.

We say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if $Output(\mathcal{C}) = \texttt{yes}$ (resp., $Output(\mathcal{C}) = \texttt{no}$), that is, if object `yes` (resp., object `no`) appears in the output region associated with the corresponding halting configuration of $\mathcal{C}$, and neither object `yes` nor `no` appears in the output region associated with any non–halting configuration of $\mathcal{C}$.

Let us notice that if a recognizer P system

$$\Pi = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$$

has a symport rule of the type $(i, \lambda/u, 0)$ then $alph(u) \cap (\Gamma \setminus \mathcal{E}) \neq \emptyset$, that is, the multiset u must contains some object from $\Gamma \setminus \mathcal{E}$ because on the contrary, all computations of Π would be not halting.

For each natural number $k \geq 1$, we denote by $\mathbf{CDC}(k)$ (respectively, $\mathbf{CDS}(k)$ or $\mathbf{CDA}(k)$) the class of recognizer P systems with membrane division and with symport/antiport rules (resp., allowing only symport or antiport rules) of length at most k. In the case of P systems without environment, we denote by $\widehat{\mathbf{CDC}}(k)$ ($\widehat{\mathbf{CDS}}(k)$ or $\widehat{\mathbf{CDA}}(k)$ respectively) the class of recognizer P systems with membrane division without environment and with symport/antiport rules (allowing only symport or only antiport rules respectively) of length at most k.

4 Polynomial Complexity Classes of P Systems with Symport/Antiport

In this section, we define what solving a decision problem in the framework of P systems with symport/antiport rules in a uniform and efficient way, means. Bearing in mind that they provide devices with a finite description, a numerable family of membrane systems will be necessary in order to solve a decision problem.

Definition 5. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in a uniform way and polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer P systems with symport/antiport rules and membrane division (with or without environment) if the following holds:*

- *The family* $\mathbf{\Pi}$ *is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(n)$ *from* $n \in \mathbb{N}$.
- *There exists a pair* (cod, s) *of polynomial-time computable functions over* I_X *such that:*
  - *for each instance* $u \in I_X$, $s(u)$ *is a natural number, and* $cod(u)$ *is an input multiset of the system* $\Pi(s(u))$;
  - *for each* $n \in \mathbb{N}$, $s^{-1}(n)$ *is a finite set;*
  - *the family* $\mathbf{\Pi}$ *is* polynomially bounded *with regard to* (X, cod, s), *that is, there exists a polynomial function* p, *such that for each* $u \in I_X$ *every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is halting and it performs at most* $p(|u|)$ *steps;*
  - *the family* $\mathbf{\Pi}$ *is* sound *with regard to* (X, cod, s), *that is, for each* $u \in I_X$, *if there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$, *then* $\theta_X(u) = 1$;
  - *the family* $\mathbf{\Pi}$ *is* complete *with regard to* (X, cod, s), *that is, for each* $u \in I_X$, *if* $\theta_X(u) = 1$, *then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

From the soundness and completeness conditions above we deduce that every P system $\Pi(n)$ is *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

Let $\mathbf{R}$ be a class of recognizer P systems with symport/antiport rules. We denote by $\mathbf{PMC_R}$ the set of all decision problems which can be solved in a uniform way and polynomial time by means of families of systems from $\mathbf{R}$. The class $\mathbf{PMC_R}$ is closed under complement and polynomial–time reductions [9].

In what follows, we prove two technical results concerning recognizer P systems.

Proposition 1. *Let* $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$ *be a recognizer P systems with symport/antiport rules with length at most* k, $k \geq 2$, *and without membrane division. Let* $M = |\mathcal{M}_1 + \cdots + \mathcal{M}_q|$ *and let* $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_m)$ *be a computation of* Π *Then,* $|\mathcal{C}_0^*| = M$, *and for each* t, $0 \leq t < m$, *we have*

$$|\mathcal{C}_{t+1}^*| \leq |\mathcal{C}_t^*| \cdot k, \text{ and } |\mathcal{C}_{t+1}^*| \leq M \cdot k^t$$

Proof: Obviously, $|\mathcal{C}_0^*| = |\mathcal{C}_0(0) + \mathcal{C}_0(1) + \cdots + \mathcal{C}_0(q)| = |\mathcal{M}_1 + \cdots + \mathcal{M}_q| = M$. Suppose $0 \leq t < m$, and let us compute $\mathcal{C}_{t+1}^* = \mathcal{C}_{t+1}(0) + \mathcal{C}_{t+1}(1) + \cdots + \mathcal{C}_{t+1}(q)$. Bearing in mind that only the skin membrane can send and receive objects from the environment, we have

$$\mathcal{C}_{t+1}(0) + \mathcal{C}_{t+1}(2) + \mathcal{C}_{t+1}(3) + \cdots + \mathcal{C}_{t+1}(q) \subseteq \mathcal{C}_t(0) + \mathcal{C}_t(1) + \cdots + \mathcal{C}_t(q)$$

Next, let us see what objects membrane 1 can receive at step $t + 1$.

- On the one hand, membrane 1 can receive objects from $\mathcal{C}_t(0)$.
- On the other hand, membrane 1 can receive objects from $\mathcal{E}$ by means of rules in the skin membrane of the types:

- $(a\ e_{i_1} \dots\ e_{i_r}, in)$ with $a \in \mathcal{C}_t(0)$ and $e_{i_1}, \dots, e_{i_r} \in \mathcal{E}$, $r \leq k-1$.
- $(a, out; e_{i_1} \dots\ e_{i_r}, in)$ with $a \in \mathcal{C}_t(1)$ and $e_{i_1}, \dots, e_{i_r} \in \mathcal{E}$, $r \leq k-1$.

Then, $|\mathcal{C}_{t+1}(1)| \leq |\mathcal{C}_t(0) + \mathcal{C}_t(1)| \cdot (k-1)$. So, we have

$$\begin{aligned} |\mathcal{C}^*_{t+1}| &= |\mathcal{C}_{t+1}(0) + \mathcal{C}_{t+1}(2) + \mathcal{C}_{t+1}(3) + \cdots + \mathcal{C}_{t+1}(q)| + |\mathcal{C}_{t+1}(1)| \\ &\leq |\mathcal{C}_t(0) + \mathcal{C}_t(1) + \cdots + \mathcal{C}_t(q)| + |\mathcal{C}_t(0) + \mathcal{C}_t(1)| \cdot (k-1) \\ &\leq |\mathcal{C}^*_t| + |\mathcal{C}^*_t| \cdot (k-1) \leq |\mathcal{C}^*_t| \cdot k \end{aligned}$$

Finally, let us see that $|\mathcal{C}^*_{t+1}| \leq M \cdot k^t$ by induction on t. For $t = 1$ the result is trivial because of $|\mathcal{C}^*_1| \leq (|\mathcal{C}^*_0| + M) \cdot (k-1) = 2M \cdot (k-1)$.

Let t be such that $1 < t < m$ and the result holds for t. Then,

$$|\mathcal{C}^*_{t+1}| \leq |\mathcal{C}^*_t| \cdot k \overset{h.i}{\leq} M \cdot k^{t-1} \cdot k = M \cdot k^t$$

□

Proposition 2. *Let* $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ *a family of recognizer P systems from* $\mathbf{CDC}(k)$*, where* $k \geq 2$*, solving a decision problem* $X = (I_X, \theta_X)$ *in polynomial time according to Definition 5. Let* (cod, s) *be a polynomial encoding associated with that solution. There exists a polynomial function* $r(n)$ *such that for each instance* $u \in I_X$*,* $2^{r(|u|)}$ *is an upper bound of the number of objects in all membranes of the system* $\Pi(s(u)) + cod(u)$ *along any computation.*

Proof: Let $p(n)$ be a polynomial function such that for each $u \in I_X$ every computation of $\Pi(s(u)) + cod(u)$ is halting and it performs at most $p(|u|)$ steps.

Let $u \in I_X$ be an instance of X and

$$\Pi(s(u)) + cod(u) = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}_1, \dots, \mathcal{R}_q, i_{in}, i_{out})$$

Let $M = |\mathcal{M}_1 + \cdots + \mathcal{M}_q|$. Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_m)$, $0 \leq m \leq p(|u|)$, be a computation of Π.

First, let us suppose that we apply only communication rules at m consecutive transition steps. From Proposition 1 we deduce that $|\mathcal{C}^*_0| = M$ and $|\mathcal{C}^*_{t+1}| \leq M \cdot k^t$, for each t, $0 \leq t < m$. Thus, if we apply in a consecutive way the maximum possible number of communication rules (without applying any division rules) to the system $\Pi(s(u)) + cod(u)$, in any instant of any computation of the system, $M \cdot k^{p(|u|)}$ is an upper bound of the number of objects in the whole system.

Now, let us consider the effect of applying in a consecutive way the maximum possible number of division rules (without applying any communication rules) to the system $\Pi(s(u)) + cod(u)$ when the initial configuration has $M \cdot k^{p(|u|)}$ objects. After that, an upper bound of the number of objects in the whole system by any computation is $M \cdot k^{p(|u|)} \cdot 2^{p(|u|)} \cdot p(|u|)$. Then, we consider a polynomial function $r(n)$ such that $r(|u|) \geq \log(M) + p(|u|) \cdot (1 + \log k) + \log(p(|u|))$, for each instance $u \in I_X$. The polynomial function $r(n)$ fulfills the property required.

□

Corollary 1. *Let* $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ *a family of recognizer P systems with symport/antiport rules and membrane division, solving a decision problem* $X = (I_X, \theta_X)$ *in polynomial time according to Definition 5. Let* (cod, s) *a polynomial encoding associated with that solution. Then, there exists a polynomial function* $r(n)$ *such that for each instance* $u \in I_X$, $2^{r(|u|)}$ *is an upper bound of the number of objects from* $\mathcal{E}$ *which are moved from the environment to all membranes of the system* $\Pi(s(u)) + cod(u)$ *along any computation.*

Proof: It suffices to note that from Proposition 2 there exists a polynomial function $r(n)$ such that for each instance $u \in I_X$, $2^{r(|u|)}$ is an upper bound of the number of objects in all membranes of the system $\Pi(s(u)) + cod(u)$. □

5 Simulating Systems from CDC(k) by Means of Systems from $\widehat{\text{CDC}}(k)$

The goal of this section is to show that any P system with symport/antiport rules and membrane division can be simulated by a P system symport/antiport rules, membrane division and without environment, in an efficient way.

First of all, we define the meaning of efficient simulations in the framework of recognizer P systems with symport/antiport rules.

Definition 6. *Let* Π *and* Π' *be recognizer P systems with symport/antiport rules. We say that* Π' simulates Π in an efficient way *if the following holds:*

1. Π' *can be constructed from* Π *by a deterministic Turing machine working in polynomial time.*
2. *There exists an injective function,* f*, from the set* $\mathbf{Comp}(\Pi)$ *of computations of* Π *onto the set* $\mathbf{Comp}(\Pi')$ *of computations of* Π' *such that:*
   - ⋆ *There exists a deterministic Turing machine that constructs computation* $f(\mathcal{C})$ *from computation* $\mathcal{C}$ *in polynomial time.*
   - ⋆ *A computation* $\mathcal{C} \in \mathbf{Comp}(\Pi)$ *is an accepting computation if and only if* $f(\mathcal{C}) \in \mathbf{Comp}(\Pi')$ *is an accepting one.*
   - ⋆ *There exists a polynomial function* $p(n)$ *such that for each* $\mathcal{C} \in \mathbf{Comp}(\Pi)$ *we have* $|f(\mathcal{C})| \leq p(|\mathcal{C}|)$.

Now, for every family of recognizer P system with symport/antiport rules and membrane division solving a decision problem, we design a family of recognizer P systems with symport/antiport rules, membrane division and *without environment* efficiently simulating it, according to Definition 6.

In what follows throughout this Section, let $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ a family of recognizer P systems with symport/antiport rules and membrane division solving a decision problem $X = (I_X, \theta_X)$ in polynomial time according to Definition 5, and let $r(n)$ be a polynomial function such that for each instance $u \in I_X$, $2^{r(|u|)}$ is an upper bound of the number of objects from $\mathcal{E}$ which are moved from the environment to all membranes of the system by any computation of $\Pi(s(u)) + cod(u)$.

Definition 7. *For each $n \in \mathbb{N}$, let*

$$\Pi(n) = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \ldots, \mathcal{R}_q, i_{in}, i_{out})$$

an element of the previous family $\mathbf{\Pi}$*, and for the sake of simplicity we denote r instead of $r(n)$ and 1 is the label of the skin membrane. Let us consider the recognizer P system with symport/antiport rules of degree $q_1 = 1 + q \cdot (r+2) + |\mathcal{E}|$, with membrane division and without environment*

$$\mathbf{S}(\Pi(n)) = (\Gamma', \Sigma', \mu', \mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_{q_1}, \mathcal{R}'_0, \mathcal{R}'_1, \ldots, \mathcal{R}'_{q_1}, i'_{in}, i'_{out})$$

defined as follows:

- $\Gamma' = \Gamma \ \cup \ \{\alpha_i : \ 0 \leq i \leq r-1\}$.
- $\Sigma' = \Sigma$.
- *Each membrane $i \in \{1, \ldots, q\}$ of Π provides a membrane of $\mathbf{S}(\Pi(n))$ with the same label. In addition, $\mathbf{S}(\Pi(n))$ has:*
  - ⋆ *$r+1$ new membranes, labelled by $(i,0), (i,1), \ldots, (i,r)$, respectively, for each $i \in \{1, \ldots, q\}$.*
  - ⋆ *A distinguished membrane labelled by 0.*
  - ⋆ *A new membrane, labelled by l_b, for each $b \in \mathcal{E}$.*
- *μ' is the rooted tree obtained from μ as follows:*
  - ⋆ *Membrane 0 is the root of μ' and it is the father of the root of μ.*
  - ⋆ *For each $b \in \mathcal{E}$, membrane 0 is the father of membrane l_b.*
  - ⋆ *We consider a linear structure whose nodes are $(i,0), (i,1), \ldots, (i,r)$ and such that (i,j) is the father of $(i,j-1)$, for each $1 \leq i \leq q$ and $1 \leq j \leq r$.*
  - ⋆ *For each membrane i of μ we add the previous linear structure being membrane i the father of membrane (i,r).*
- *Initial multisets: $\mathcal{M}'_0 = \emptyset$, $\mathcal{M}'_{l_b} = \{\alpha_0\}$, for each $b \in \mathcal{E}$, and*

$$(1 \leq i \leq q) \begin{cases} \mathcal{M}'_{(i,0)} = \mathcal{M}_i \\ \mathcal{M}'_{(i,1)} = \ \emptyset \\ \ldots \qquad \ldots \\ \mathcal{M}'_{(i,r)} = \ \emptyset \\ \mathcal{M}'_i \quad = \ \emptyset \end{cases}$$

- *Set of rules:*

$$\mathcal{R}'_0 \cup \mathcal{R}'_1 \cup \cdots \cup \mathcal{R}'_q \cup \{\mathcal{R}'_{(i,j)} : 1 \leq i \leq q, 0 \leq j \leq r\} \cup \{\mathcal{R}'_{l_b} : b \in \mathcal{E}\}$$

where $\mathcal{R}'_0 = \emptyset$, $\mathcal{R}'_i = \mathcal{R}_i$ for $1 \leq i \leq q$, and

$$\begin{array}{rl} \mathcal{R}'_{(i,j)} & = \{(a, out; \lambda, in) : \ a \in \Gamma\}, \ \text{for } 1 \leq i \leq q \ \wedge \ 0 \leq j \leq r\} \\ \mathcal{R}'_{l_b} & = \{[\alpha_j]_{l_b} \to [\alpha_{j+1}]_{l_b} \ [\alpha_{j+1}]_{l_b} : \ 0 \leq j \leq r-2\} \cup \\ & \quad \{[\alpha_{r-1}]_{l_b} \to [b]_{l_b} \ [b]_{l_b}, (l_b, out; \lambda, in)\}, \ \text{for } b \in \mathcal{E} \end{array}$$

- $i'_{in} = (i_{in}, 0)$, *and* $i'_{out} = 0$.

Let us notice that $\mathbf{S}(\Pi(n))$ can be considered as an extension of $\Pi(n)$ without environment, in the following sense:

- $\star$ $\Gamma \subseteq \Gamma', \Sigma \subseteq \Sigma'$ and $\mathcal{E} = \emptyset$.
- $\star$ Each membrane in Π is also a membrane in $\mathbf{S}(\Pi(n))$.
- $\star$ There is a distinguished membrane in $\mathbf{S}(\Pi(n))$ labelled by 0 which plays the role of environment of $\Pi(n)$.
- $\star$ μ is a subtree of μ'.
- $\star$ $\mathcal{R} \subseteq \mathcal{R}'$, and now 0 is the label of a "ordinary membrane" in $\mathbf{S}(\Pi(n))$.

Next, we analyze the structure of the computations of system $\mathbf{S}(\Pi(n))$ and we compare them with the computations of $\Pi(n)$.

Lemma 1. *Let $\mathcal{C}' = (\mathcal{C}'_0, \mathcal{C}'_1, \dots)$ be a computation of $\mathbf{S}(\Pi(n))$. For each t $(1 \leq t \leq r)$ the following holds:*

- *$\mathcal{C}'_t(i) = \emptyset$, for $0 \leq i \leq q$.*
- *For each $1 \leq i \leq q$, and $0 \leq j \leq r$ we have:*

$$\mathcal{C}'_t(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = t \\ \emptyset, \text{ if } & j \neq t \end{cases}$$

- *For each $b \in \mathcal{E}$, there exist 2^t membranes labelled by l_b whose father is membrane 0 and their content is:*

$$\mathcal{C}'_t(l_b) = \begin{cases} \{\alpha_t\}, \text{ if } & 1 \leq t \leq r-1 \\ \{b\}, \text{ if } & t = r \end{cases}$$

Proof: By induction on t.

Let us start with the basic case $t = 1$. The initial configuration of system $\mathbf{S}(\Pi(n))$ is the following:

- $\mathcal{C}'_0(i) = \emptyset$, for $0 \leq i \leq q$.
- For each $1 \leq i \leq q$ we have $\mathcal{C}'_0(i,0) = \mathcal{M}_i$, and $\mathcal{C}'_0(i,j) = \emptyset$, for $1 \leq j \leq r$.
- For each $b \in \mathcal{E}$, there exists only one membrane labelled by l_b whose contents is $\{\alpha_0\}$.

At configuration $\mathcal{C}'_0$, only the following rules are applicable:

- $[\alpha_0]_{l_b} \to [\alpha_1]_{l_b}\ [\alpha_1]_{l_b}$, for each $b \in \mathcal{E}$.
- $\big(a, out; \lambda, in\big) \in \mathcal{R}_{(i,0)}$, for each $a \in supp(\mathcal{M}_i)$.

Thus,

(a) For each i $(1 \leq i \leq q)$ we have:

$$\begin{cases} \mathcal{C}'_1(i) & = \emptyset \\ \mathcal{C}'_1(0) & = \emptyset \\ \mathcal{C}'_1(i,0) & = \emptyset \\ \mathcal{C}'_1(i,1) & = \mathcal{M}_i \\ \mathcal{C}'_1(i,j) & = \emptyset, \text{ for } 2 \leq j \leq r \end{cases}$$

(b) For each $b \in \mathcal{E}$, there are 2 membranes labelled by l_b whose father is membrane 0 and their content is $\{\alpha_1\}$.

Hence, the result holds for $t = 1$.

By induction hypothesis, let t be such that $1 \leq t < r$, and let us suppose the result holds for t, that is,

- $\mathcal{C}'_t(i) = \emptyset$, for $0 \leq i \leq q$.
- For each $1 \leq i \leq q$, and $0 \leq j \leq r$ we have:

$$\mathcal{C}'_t(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = t \\ \emptyset, \text{ if } & j \neq t \end{cases}$$

- For each $b \in \mathcal{E}$, there exist 2^t membranes labelled by l_b whose father is membrane 0 and their content is $\mathcal{C}'_t(l_b) = \{\alpha_t\}$ (because $t \leq r-1$).

Then, at configuration $\mathcal{C}'_t$ only the following rules are applicable:

(1) If $t \leq r-2$, the rules $[\alpha_t]_{l_b} \rightarrow [\alpha_{t+1}]_{l_b}\ [\alpha_{t+1}]_{l_b}$, for each $b \in \mathcal{E}$.
(2) If $t = r-1$, the rules $[\alpha_t]_{l_b} \rightarrow [b]_{l_b}\ [b]_{l_b}$, for each $b \in \mathcal{E}$.
(3) $\big(a, out; \lambda, in\big) \in \mathcal{R}_{(i,t)}$, for each $a \in supp(\mathcal{M}_i)$.

From the application of rules of types (1) or (2) at configuration $\mathcal{C}'_t$, we deduce that there are 2^{t+1} membranes labelled by l_b in $\mathcal{C}'_{t+1}$, for each $b \in \mathcal{E}$, whose father is membrane 0 and their content is $\{\alpha_{t+1}\}$, if $t \leq r-2$, or $\{b\}$, if $t = r-1$.

From the application of rules of type (3) at configuration $\mathcal{C}'_t$, we deduce that

$$\mathcal{C}'_{t+1}(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = t+1 \\ \emptyset, \text{ if } & 0 \leq j \leq r \ \wedge\ j \neq t+1 \end{cases}$$

Bearing in mind that no other rule of system $\mathbf{S}(\Pi(n))$ is applicable, we deduce that $\mathcal{C}'_{t+1}(i) = \emptyset$, for $0 \leq i \leq q$.

This completes the proof of this Lemma.

□

Lemma 2. *Let $\mathcal{C}' = (\mathcal{C}'_0, \mathcal{C}'_1, \dots)$ be a computation of the P system $\mathbf{S}(\Pi(n))$. Configuration $\mathcal{C}'_{r+1}$ is the following:*

(1) $\mathcal{C}'_{r+1}(0) = b_1^{2^r} \dots b_\alpha^{2^r}$, where $\mathcal{E} = \{b_1, \dots, b_\alpha\}$.
(2) $\mathcal{C}'_{r+1}(i) = \mathcal{M}_i = \mathcal{C}_0(i)$, for $1 \leq i \leq q$.
(3) $\mathcal{C}'_{r+1}(i,j) = \emptyset$, for $1 \leq i \leq q$, $0 \leq j \leq r$.
(4) For each $b \in \mathcal{E}$, there exist 2^r membranes labelled by l_b whose father is membrane 0 and their content is empty.

Proof: From Lemma 1, the configuration $\mathcal{C}'_r$ is the following:

- $\mathcal{C}'_r(i) = \emptyset$, for $0 \leq i \leq q$.
- For each i $(1 \leq i \leq q)$ we have

$$\mathcal{C}'_r(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = r \\ \emptyset, \text{ if } & j \neq r \end{cases}$$

- For each $b \in \mathcal{E}$, there exist 2^r membranes labelled by l_b whose father is membrane 0 and their content is $\{b\}$.

At configuration $\mathcal{C}'_r$ only the following rules are applicables:

- $\big(a, out; \lambda, in\big) \in \mathcal{R}_{(i,r)}$, for each $a \in \Gamma \cap supp(\mathcal{M}_i)$.
- $\big(b, out; \lambda, in\big) \in \mathcal{R}_{l_b}$, for each $b \in \mathcal{E}$.

Thus,

- $\mathcal{C}'_{r+1}(0) = b_1^{2^r} \dots b_\alpha^{2^r}$, where $\mathcal{E} = \{b_1, \dots, b_\alpha\}$.
- $\mathcal{C}'_{r+1}(i) = \mathcal{M}_i = \mathcal{C}_0(i)$, for $1 \leq i \leq q$.
- $\mathcal{C}'_{r+1}(i,j) = \emptyset$, for $1 \leq i \leq q$ and $0 \leq j \leq r$.
- For each $b \in \mathcal{E}$, there exist 2^r membranes labelled by l_b whose father is membrane 0 and their content is empty.

□

Definition 8. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_m)$ be a halting computation of $\Pi(n)$. Then we define the computation $\mathbf{S}(\mathcal{C}) = (\mathcal{C}'_0, \mathcal{C}'_1, \dots, \mathcal{C}'_r, \mathcal{C}'_{r+1}, \dots, \mathcal{C}'_{r+1+m})$ of $\mathbf{S}(\Pi(n))$ as follows:*

(1) *The initial configuration is:*
$$\begin{cases} \mathcal{C}'_0(i) & = \emptyset, \text{ for } 0 \leq i \leq q \\ \mathcal{C}'_0(i,0) & = \mathcal{C}_0(i), \text{ for } 1 \leq i \leq q \\ \mathcal{C}'_0(i,j) & = \emptyset, \text{ for } 1 \leq i \leq q \text{ and } 1 \leq j \leq r \\ \mathcal{C}'_0(l_b) & = \alpha_0, \text{ for each } b \in \mathcal{E} \end{cases}$$
(2) *The configuration $\mathcal{C}'_t$, for $1 \leq t \leq r$, is described by Lemma 1.*
(3) *The configuration $\mathcal{C}'_{r+1}$ is described by Lemma 2.*
(4) *The configuration $\mathcal{C}'_{r+1+s}$, for $0 \leq s \leq m$, coincides with the configuration $\mathcal{C}_s$ of Π, that is, $\mathcal{C}_s(i) = \mathcal{C}'_{r+1+s}(i)$, for $1 \leq i \leq q$. The content of the remaining membranes (excluding membrane 0) at configuration $\mathcal{C}'_{r+1+s}$ is equal to the content of that membrane at configuration $\mathcal{C}'_{r+1}$, that is, these membranes do not evolve after step $r+1$.*

That is, every computation $\mathcal{C}$ of $\Pi(n)$ can be "reproduced" by a computation $\mathbf{S}(\mathcal{C})$ of $\mathbf{S}(\Pi(n))$ with a delay: from step $r+1$ to step $r+1+m$, the computation $\mathbf{S}(\mathcal{C})$ restricted to membranes $1, \dots, q$ provides the computation $\mathcal{C}$ of $\Pi(n)$.

From Lemma 1 and Lemma 2 we deduce the following:

(a) $\mathbf{S}(\mathcal{C})$ is a computation of $\mathbf{S}(\Pi(n))$.
(b) $\mathbf{S}$ is an injective function from $\mathbf{Comp}(\Pi(n))$ onto $\mathbf{Comp}(S(\Pi(n)))$.

Proposition 3. *The P system $\mathbf{S}(\Pi(n))$ defined in Definition 7 simulates $\Pi(n)$ in an efficient way.*

Proof. In order to show that $\mathbf{S}(\Pi(n))$ can be constructed from $\Pi(n)$ by a deterministic Turing machine working in polynomial time, it is enough to note that the amount of resources needed to construct $\mathbf{S}(\Pi(n))$ from $\Pi(n)$ is polynomial in the size of the initial resources of $\Pi(n)$. Indeed,

1. The size of the alphabet of $\mathbf{S}(\Pi(n))$ is $|\Gamma'| = |\Gamma| + r$.
2. The initial number of membranes of $\mathbf{S}(\Pi(n))$ is $1 + q \cdot (r + 2) + |\mathcal{E}|$.
3. The initial number of objects of $\mathbf{S}(\Pi(n))$ is the initial number of objects of $\Pi(n)$ plus $|\mathcal{E}|$.
4. The number of rules of $\mathbf{S}(\Pi(n))$ is $|\mathcal{R}'| = |\mathcal{R}| + (r + 1) \cdot |\mathcal{E}| + |\Gamma| \cdot q \cdot (r + 1)$.
5. The maximal length of a communication rule of $\mathbf{S}(\Pi(n))$ is equal to the maximal length of a communication rule of $\Pi(n)$.

From Lemma 1 and Lemma 2 we deduce that:

(a) Every computation $\mathcal{C}'$ of $\mathbf{S}(\Pi(n))$ has associated a computation $\mathcal{C}$ of $\Pi(n)$ such that $\mathbf{S}(\mathcal{C}) = \mathcal{C}'$ in a natural way.
(b) The function $\mathbf{S}$ is injective.
(c) A computation $\mathcal{C}$ of $\Pi(n)$ is an accepting computation if and only if $\mathbf{S}(\mathcal{C})$ is an accepting computation of $\mathbf{S}(\Pi(n))$.

Finally, let us notice that if $\mathcal{C}$ is a computation of $\Pi(n)$ with length m, then $\mathbf{S}(\mathcal{C})$ is a computation of $\mathbf{S}(\Pi(n))$ with length $r + 1 + m$. □

6 Computational Complexity Classes of P Systems with Membrane Division and Without Environment

In this Section, we analyze the role of the environment in the efficiency of P systems with membrane division. That is, we study the ability of these P systems with respect to the computational efficiency when the alphabet of the environment is an empty set.

Theorem 1. *For each $k \in \mathbb{N}$ we have* $\mathbf{PMC}_{\mathbf{CDC}(k+1)} = \mathbf{PMC}_{\widehat{\mathbf{CDC}}(k+1)}$.

Proof: Let us recall that $\mathbf{PMC}_{\mathbf{CDC}(1)} = \mathbf{P}$ (see [4] for details). Then,

$$\mathbf{P} \subseteq \mathbf{PMC}_{\widehat{\mathbf{CDC}}(1)} \subseteq \mathbf{PMC}_{\mathbf{CDC}(1)} = \mathbf{P}$$

Thus, the result holds for $k = 0$. Let us show the result holds for $k \geq 1$. Since $\widehat{\mathbf{CDC}}(k+1) \subseteq \mathbf{CDC}(k+1)$ it suffices to prove that $\mathbf{PMC}_{\mathbf{CDC}(k+1)} \subseteq \mathbf{PMC}_{\widehat{\mathbf{CDC}}(k+1)}$. For that, let $X \in \mathbf{PMC}_{\mathbf{CDC}(\mathbf{k+1})}$.

Let $\{\Pi(n) \mid n \in N\}$ be a family of P systems from $\mathbf{CDC}(k+1)$ solving X according to Definition 5. Let (cod, s) be a polinomial encoding associated with that solution. Let $u \in I_X$ be an instance of the problem X that will be processed by the system $\Pi(s(u)) + cod(u)$. According to Proposition 2, let $r(n)$ be a polynomial function that $2^{r(|u|)}$ is an upper bound of the number of objects from $\mathcal{E}$ which are moved from the environment to all membranes of the system by any computation of

$$\Pi(s(u)) + cod(u) = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} + cod(u), \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$$

Then, we consider the P system without environment

$$\mathbf{S}(\Pi(s(u))) + cod(u) = (\Gamma', \Sigma', \mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_{i_{in}} + cod(u), \ldots, \mathcal{M}'_{q_1}, \mathcal{R}', i'_{in}, i'_{out})$$

according to Definition 7, where $q_1 = 1 + q \cdot (r(|u|) + 2) + |\mathcal{E}|$.

Therefore, $\mathbf{S}(\Pi(s(u))) + cod(u)$ is a P system from $\widehat{\mathbf{CDC}}(k+1)$ such that verifies the following:

- A distinguished membrane labelled by 0 has been considered, which will play the role of the environment at the system $\Pi(s(u)) + cod(u)$.
- At the initial configuration, it has enough objects in membrane 0 in order to simulate the behaviour of the environment of the system $\Pi(s(u))) + cod(u)$.
- After $r(n) + 1$ step, computations of $\Pi(s(u)) + cod(u)$ are reproduced by the computations of $\mathbf{S}(\Pi(s(u))) + cod(u)$ exactly.

Let us suppose that $\mathcal{E} = \{b_1, \ldots, b_\alpha\}$. In order to simulate $\Pi(s(u)) + cod(u)$ by a P system without environment in an efficient way, we need to have enough objects in the membrane of $\mathbf{S}(\Pi(s(u))) + cod(u)$ labelled by 0 available. Specifically, $2^{r(n)}$ objects in that membrane are enough.

In order to start the simulation of any computation $\mathcal{C}$ of $\Pi(s(u)) + cod(u)$, it would be enough to have $2^{r(n)}$ copies of each object $b_j \in \mathcal{E}$ in the membrane of $\mathbf{S}(\Pi(s(u))) + cod(u)$ labelled by 0. For this purpose,

- For each $b \in \mathcal{E}$ we consider a membrane in $\mathbf{S}(\Pi(s(u))) + cod(u)$ labelled by l_b which only contains object α_0 initially. We also consider the following rules:
  - $[\alpha_j]_{l_b} \to [\alpha_{j+1}]_{l_b}\ [\alpha_{j+1}]_{l_b}$, for $0 \leq j \leq r(|u|) - 2$,
  - $[\alpha_{p(n)-1}]_{l_b} \to [b]_{l_b}\ [b]_{l_b}$,
  - $(l_b, b/\lambda, 0)$.
- By applying the previous rules, after $r(|u|)$ transition steps we get $2^{r(|u|)}$ membranes labelled by l_b, for each $b \in \mathcal{E}$ in such a way that each of them contains only object b. Finally, by applying the third rule we get $2^{r(|u|)}$ copies of objects b in membrane 0, for each $b \in \mathcal{E}$.

Therefore, after the execution of $r(|u|) + 1$ transition steps in each computation of $\mathbf{S}(\Pi(s(u))) + cod(u)$ in membrane 0 of the corresponding configuration, we have $2^{r(|u|)}$ copies of each object $b \in \mathcal{E}$. This number of copies is enough to simulate any computation $\mathcal{C}$ of $\Pi(s(u)) + cod(u)$ through the system $\mathbf{S}(\Pi(s(u)) + cod(u))$.

From Proposition 3 we deduce that the family $\{\mathbf{S}(\Pi(n))|\ n \in N\}$ solves X in polynomial time according to Definition 5. Hence, $X \in \mathbf{PMC}_{\widehat{\mathbf{CDC}}(k+1)}$. □

7 Conclusions and Further Works

Initial configurations of ordinary P systems with symport/antiport rules have an arbitrarily large amount of copies of some kind of objects belonging to a distinguished alphabet which specifies the *environment* of the system.

The previous condition is no too nice from the computational complexity point of view. In this paper, we show that in P systems with with symport/antiport rules and membrane division the environment can be "removed" without a loss of efficiency.

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (61033003, 91130034 and 61320106005). The work of M.J. Pérez and L.F. Macías was supported by Project TIN2012-37434 of the Ministerio de Ciencia e Innovación of Spain.

References

1. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A. Romero-Jiménez. . Computational efficiency of cellular division in tissue-like P systems. *Romanian Journal of Information Science and Technology*, **11**, 3 (2008), 229-241.
2. Gutiérrez-Escudero, R., Pérez-Jiménez, M.J. and Rius-Font, M. Characterizing tractability by tissue-like P systems. *Lecture Notes in Computer Science* **5957**, 5957 (2010), 289-300.
3. Macías-Ramos, L.F., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rius-Font, M. and Valencia-Cabrera, L. The efficiency of tissue P systems with cell separation rely on the environment. Submitted, 2012
4. Macías-Ramos, L.F., Song B., Song T., Pan L., Pérez-Jiménez, M.J. Limits on efficient computation in P systems with symport/antiport rules. Proceedings of the Fifteenth Brainstorming Week on Membrane Computing. Submitted 2017.
5. Pan, L. and Ishdorj, T.-O. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5, (2004), 630–649.
6. Păun, Gh. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.
7. Păun, Gh. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
8. Păun, Gh., Pérez-Jiménez, M.J. and Riscos-Núñez, A. Tissue P System with cell division. *In. J. of Computers, communications & control*, **3**, 3, (2008), 295–303.
9. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
10. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, **11**, 4, (2006), 423-434.
11. Porreca, A.E., Murphy, N. Pérez-Jiménez, M.J. An efficient solution of Ham Cycle problem in tissue P systems with cell division and communication rules with length at most 2. In M. Garca-Quismondo, L.F. Macas-Ramos, Gh. Paun, I. Prez Hurtado, L. Valencia-Cabrera (eds.) *Proceedings of the Tenth Brainstorming Week on Membrane*

Computing, Volume II, Seville, Spain, January 30- February 3, 2012, Report RGNC 01/2012, Fnix Editora, 2012, pp. 141-166.

Limits on Efficient Computation in P Systems with Symport/Antiport Rules

Luis F. Macías-Ramos[1], Bosheng Song[2],
Tao Song[2], Linqiang Pan[2], Mario J. Pérez-Jiménez[1]

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

[2] Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China
E-mail: boshengsong@163.com, songtao0608@hotmail.com,
lqpan@mail.hust.edu.cn

Summary. Classical membrane systems with symport/antiport rules observe the *conservation law*, in the sense that they compute by changing the places of objects with respect to the membranes, and not by changing the objects themselves. In these systems the environment plays an active role because the systems not only send objects to the environment, but also bring objects from the environment. In the initial configuration of a system, there is a special alphabet whose elements appear in an arbitrary large number of copies. The ability of these computing devices with infinite copies of some objects has been widely exploited in the design of efficient solutions to computationally hard problems. This paper deals with computational aspects of P systems with symport/antiport rules and membrane division rules or membrane separation rules. Specifically, we study the limitations of such P systems when the only communication rules allowed have length 1.

Key words: Membrane Computing, P System with Symport/Antiport rules, Membrane Division, Membrane Separation, Computational Complexity.

1 Introduction

In Chapter 3, the computation efficiency of membrane systems has been studied and new techniques and tools have been provided to tackle the **P** versus **NP** problem. For that, two framework has been considered: cell-like P systems with active membranes (with or without using electrical charges) and tissue-like P systems

with cell division or cell separation. In both cases, the communication rules are different. In the case of cell-like P systems, evolution rules, send-in and send-out rules and dissolution rules are considered. In the case of tissue-like P systems, communication rules have been implemented by using symport/antiport rules.

Membrane computing is a flexible and versatile branch of natural computing, which arises as an abstraction of the compartmentalized structure of living cells, and the way biochemical substances are processed in (or moved between) membrane bounded regions [10]. Inspired by the structure of living cells, two main classes of membrane systems have been investigated: a hierarchical (cell-like) arrangement of membranes, inspired from the structure of the cell [10] and a net of membranes (placed in the nodes of a directed graph), inspired from the cell-interconnection in tissues [5] or inspired from the the way that neurons communicate with each other by means of short electrical impulses (spikes), emitted at precise moments of time [4]. All classes of computing systems considered in the field of membrane computing are generally called *P systems*, which are parallel and distributed computational models. A comprehensive information in membrane computing can be found in [13] and [2], and for the most up-to-date source of this area, please refer to the P systems website `http://ppage.psystems.eu`.

On the one hand, cell-like P systems with symport/antiport rules were introduced in [9] aiming to abstract the biological phenomenon of trans-membrane transport of couples of chemical substances, in the same or in opposite directions. On the other hand, tissue P systems with symport/antiport rules were introduced in [8] by abstracting networks of elementary membranes such that some of them are linked by "communication channels".

In eukaryotic cells there are two relevant processes: *mitosis* and *membrane fission*. The first one is a process of nuclear division in eukaryotic cells during which one cell gives place to two genetically identical children cells. *Membrane fission* occurs when a membrane gives place to two separated membranes, that is, whenever a vesicle is produced or a larger subcellular compartment is divided into smaller discrete units. These processes have been a source of inspiration to incorporate new ingredients in membrane computing in order to be able to produce exponential workspace in polynomial time. With respect to the mitosis process, *P systems with membrane division* were introduced in [11], and with respect to the membrane fission process, *P systems with membrane separation* were introduced in [6]. These concepts were also introduced in the framework of tissue P systems: tissue P systems with cell division [12] and tissue P systems with cell separation [7].

Taking inspiration from living cells, we add abstractions of the mitosis and the membrane fission processes as ingredients in P systems with symport/antiport rules. Specifically, we allow new types of rules (membrane division and membrane separation) in that framework leading to P systems with symport/antiport rules and membrane division or membrane separation. The limitations of these systems from the efficiency point of view are studied.

The paper is structured as follows. First, some basic concepts and notations are introduced in order to provide a self-contained paper. Section 3 is devoted to define the framework of cell-like P systems with symport/antiport rules and membrane division or membrane separation. Next, recognizer tissue P systems are briefly described and computational complexity classes in these system are introduced. In Section 4, the limitations on the efficiency of cell-like P systems with membrane division or membrane separation which use communication rules of length one, that is, membrane systems without cooperation, are studied. Finally, some conclusions and open problems are presented.

2 Preliminaries

An *alphabet* Σ is a finite non-empty set and their elements are called *symbols*. An ordered finite sequence of symbols over Σ forms a *string* or *word*. The set of symbols occurring in a string u over Σ is denoted by $alph(u)$. The *length* of a string u, denoted by $|u|$, is the number of occurrences of symbols it contains. For an alphabet Σ, we denote by Σ^* the set of all strings of symbols from Σ. The empty string (with length 0) is denoted by λ. A *language* over Σ is a subset of Σ^*.

A *multiset* over an alphabet Σ, is an ordered pair (Σ, f) where $f : \Sigma \to \mathbb{N}$ is a mapping from Σ onto the set of non-negative numbers $\mathbb{N}$. If $m = (\Sigma, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in \Sigma \mid f(x) > 0\}$. A multiset is finite if its support is a finite set. We denote by $\emptyset$ the empty multiset and we denote by $\mathcal{M}_f(\Sigma)$ the set of all finite multisets over Σ.

Let $m_1 = (\Sigma, f_1)$, $m_2 = (\Sigma, f_2)$ are multisets over Σ, then the union of m_1 and m_2, denoted by $m_1 + m_2$, is the multiset (Σ, g), where $g(x) = f_1(x) + f_2(x)$ for each $x \in \Sigma$. The relative complement of m_2 in m_1, denoted by $m_1 \setminus m_2$, is the multiset (Σ, g), where $g(x) = f_1(x) - f_2(x)$ if $f_1(x) \geq f_2(x)$, and $g(x) = 0$ otherwise.

Let us recall that a *free tree* (*tree*, for short) is a connected, acyclic, undirected graph. A *rooted tree* is a tree in which one of the vertices (called *the root of the tree*) is distinguished from the others. In a rooted tree the concepts of ascendants and descendants are defined in a usual way. Given a node x (different from the root), if the last edge on the (unique) path from the root of the tree to the node x is $\{x, y\}$ (in this case, $x \neq y$), then y is **the** *parent* of node x and x is **a** *child* of node y. The root is the only node in the tree with no parent (see [1] for details).

Let us recall that the `Reachability Problem` is the following: *given a (directed or undirected) graph G and two nodes a, b, determine whether or not the node b is reachable from a, that is, whether or not there exists a path in the graph from a to b.* We denote by `Reachability`(G, a, b) the answer (`yes` or `no`) to the Reachability problem with instance (G, a, b). It is easy to design an algorithm running in polynomial time solving this problem. For example, given a (directed or undirected) graph G and two nodes a, b, we consider a depth–first–search with source a, and we check if b is in the tree of the computation forest whose root is

a. The total running time of this algorithm is $O(|V| + |E|)$, that is, in the worst case is quadratic in the number of nodes. Moreover, this algorithm needs to store a linear number of items (it can be proved that there exists another polynomial time algorithm which uses $O(\log^2(|V|))$ space).

3 P Systems with Symport/Antiport Rules

In this section we introduce a kind of cell-like P systems that use communication rules capturing the biological phenomenon of trans-membrane transports of several chemical substances. Specifically, two processes have been considered. The first one allows a multiset of chemical substances to pass through a membrane in the same direction. In the second one, two multisets of chemical substances (located in different biological membranes) only pass with the help of each other (an *exchange* of objects between both membranes).

Next, we introduce an abstraction of these operation in the framework of P systems with symport/antiport rules following [9]. In these models, the membranes are not polarized.

Definition 1. *A P system with symport/antiport rules of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$, where:*

1. *Γ is a finite* alphabet*;*
2. *$\mathcal{E} \subsetneq \Gamma$;*
3. *μ is a membrane structure (a rooted tree) whose nodes are injectively labelled with $1, 2 \ldots, q$ (the root of the tree is labelled by 1);*
4. *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multises over Γ.*
5. *$\mathcal{R}_1, \cdots, \mathcal{R}_q$ are finite set of communication rules of the following forms:*
   - ⋆ Symport rules*: (u, out) or (u, in), where u is a finite multiset over Γ such that $|u| > 0$;*
   - ⋆ Antiport rules*: $(u, out; v, in)$, where u, v are finite multisets over Γ such that $|u| > 0$ and $|v| > 0$;*
6. *$i_{out} \in \{0, 1, \ldots, q\}$.*

A P system with symport/antiport rules of degree q

$$\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$$

can be viewed as a set of q membranes, labelled by $1, \ldots, q$, arranged in a hierarchical structure μ given by a rooted tree whose root is called the *skin membrane*, such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ represent the finite multisets of objects initially placed in the q membranes of the system; (b) $\mathcal{E}$ is the set of objects initially located in the environment of the system, all of them available in an arbitrary number of copies; (c) $\mathcal{R}_1, \cdots, \mathcal{R}_q$ are finite sets of communication rules over Γ ($\mathcal{R}_i$ is associated with the membrane i of μ); and (d) i_{out} represents a distinguished *region* which will encode the output of the system. We use the term *region* i ($0 \leq i \leq q$) to refer

to membrane i in the case $1 \leq i \leq q$ and to refer to the environment in the case $i = 0$. The length of rule (u, out) or (u, in) (resp. $(u, out; v, in)$) is defined as $|u|$ (resp. $|u| + |v|$).

For each membrane $i \in \{2, \ldots, q\}$ (different from the skin membrane) we denote by $p(i)$ the parent of membrane i in the rooted tree μ. We define $p(1) = 0$, that is, by convention the "parent" of the skin membrane is the environment.

An *instantaneous description* or a *configuration* at an instant t of a P system with symport/antiport rules is described by the membrane structure at instant t, all multisets of objects over Γ associated with all the membranes present in the system, and the multiset of objects over $\Gamma - \mathcal{E}$ associated with the environment at that moment. Recall that there are infinite copies of objects from $\mathcal{E}$ in the environment, and hence this set is not properly changed along the computation. The *initial configuration* of the system is $(\mu, \mathcal{M}_1, \cdots, \mathcal{M}_q; \emptyset)$.

A symport rule $(u, out) \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if membrane i is in $\mathcal{C}_t$ and multiset u is contained in such membrane. When applying a rule $(u, out) \in \mathcal{R}_i$, the objects specified by u are sent out of membrane i into the region immediately outside (the parent $p(i)$ of i), this can be the environment in the case of the skin membrane.

A symport rule $(u, in) \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if membrane i is in $\mathcal{C}_t$ and multiset u is contained in the parent of i. When applying a rule $(u, in) \in \mathcal{R}_i$, the multiset of objects u goes out from the parent membrane of i and enters into the region defined by the membrane i.

An antiport rule $(u, out; v, in) \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if membrane i is in $\mathcal{C}_t$ and multiset u is contained in such membrane, and multiset v is contained in the parent of i. When applying a rule $(u, out; v, in) \in \mathcal{R}_i$, the objects specified by u are sent out of membrane i into the parent of i and, at the same time, bringing the objects specified by v into membrane i.

The rules of a P system with symport/antiport rules are applied in a nondeterministic maximally parallel manner: at each step we apply a multiset of rules which is maximal, no further applicable rule can be added.

Let us fix a P system with symport/antiport rules Π. We say that configuration $\mathcal{C}^1$ yields configuration $\mathcal{C}^2$ in one *transition step*, denoted by $\mathcal{C}^1 \Rightarrow_\Pi \mathcal{C}^2$, if we can pass from $\mathcal{C}^1$ to $\mathcal{C}^2$ by applying the rules from $\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_q$ following the previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that: (a) the first term of the sequence is the initial configuration of the system; (b) each non-initial configuration of the sequence is obtained from the previous configuration by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and (c) if the sequence is finite (called *halting computation*) then the last term of the sequence is a *halting configuration* (a configuration where no rule of the system is applicable to it).

All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region i_{out} in the halting configuration. If $\mathcal{C} = \{\mathcal{C}_t\}_{t \leq r}$ of Π ($r \in \mathbb{N}$) is a halting computation, then the *length of* $\mathcal{C}$, denoted by $|\mathcal{C}|$, is r, that is,

$|\mathcal{C}|$ is the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. We denote by $\mathcal{C}_t(i), 1 \leq i \leq q$, the multiset of objects over Γ contained in the membrane labelled by i at configuration $\mathcal{C}_t$. We also denote by $\mathcal{C}_t(0)$ the multiset of objects over $\Gamma \setminus \mathcal{E}$ contained in the environment at configuration $\mathcal{C}_t$.

3.1 Recognizer P systems with symport/antiport rules

Recognizer P systems were introduced in [16] and they provide a natural framework to solve decision problems. Next, we introduce the concept of recognizer associated with the systems defined in the previous section.

Definition 2. *A recognizer P system with symport/antiport rules of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$, where:*

- *$\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$ is a P system with symport/antiport rules of degree $q \geq 1$;*
- *the working alphabet Γ has two distinguished objects* `yes` *and* `no`*, with at least one copy of them presents in some initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$, but none of them present in $\mathcal{E}$;*
- *Σ is an (input) alphabet strictly contained in Γ such that $\mathcal{E} \subseteq \Gamma \setminus \Sigma$;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over $\Gamma \setminus \Sigma$;*
- *$i_{in} \in \{1, \ldots, q\}$ is the input membrane;*
- *the output region i_{out} is the environment;*
- *all computations halt;*
- *if $\mathcal{C}$ is a computation of Π, then either object* `yes` *or object* `no` *(but not both) must have been released into the environment, and only at the last step of the computation.*

Let us notice that if a recognizer P system

$$\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$$

has a symport rule of the type (u, out) or (u, in) then $alph(u) \cap (\Gamma \setminus \mathcal{E}) \neq \emptyset$, that is, the multiset u must contains some object from $\Gamma \setminus \mathcal{E}$ because on the contrary, all computations of Π would be non halting.

For each finite multiset w over the input alphabet Σ, a *computation* of $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$ with input multiset w starts from the configuration of the form $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} + w, \ldots, \mathcal{M}_q, \emptyset)$, where the input multiset w is added to the content of the input membrane i_{in}. That is, we have an initial configuration associated with each input multiset w over Σ in recognizer P systems with symport/antiport rules. We denote by $\Pi + w$ the P system Π with input multiset w.

3.2 Polynomial complexity classes of recognizer P systems with symport/antiport rules

Let us recall that a decision problem X is one whose solution is either "`yes`" or "`no`". This can be formally defined by an ordered pair (I_X, θ_X), where I_X is a language over a finite alphabet and θ_X is a total boolean function over I_X. The elements of I_X are called *instances* of the problem X. Next, according to [15], we define what solving a decision problem by a family of recognizer P systems with symport/antiport rules, *in a uniform way*, means.

Definition 3. *A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer P systems with symport/antiport rules (in a uniform way) if the following conditions hold:*

- *the family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$;*
- *there exists a pair (cod, s) of polynomial-time computable functions over I_X such that:*
  - *for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
  - *for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set;*
  - *the family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u)) + cod(u)$ is halting and it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is sound with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u)) + cod(u)$, then $\theta_X(u) = 1$;*
  - *the family $\mathbf{\Pi}$ is complete with regard to (X, cod, s), that is, for each $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u)) + cod(u)$ is an accepting one.*

According to Definition 3, we say that for each $u \in I_X$, the P system $\Pi(s(u)) + cod(u)$ is *confluent*, in the sense that all possible computations of the system must give the same answer.

If $\mathbf{R}$ is a class of recognizer P systems, then we denote by $\mathbf{PMC_R}$ the set of all decision problems which can be solved in polynomial time (and in a uniform way) by means of recognizer P systems from $\mathbf{R}$. The class $\mathbf{PMC_R}$ is closed under complement and polynomial–time reductions (see [15] for details).

3.3 P systems with symport/antiport rules and membrane division or membrane separation

In this section, we introduce new types of rules (membrane division and membrane separation) inspired by the mitosis and the membrane fission processes, in the framework of P systems with symport/antiport rules. These rules provide a mechanism to construct an exponential workspace in polynomial time.

Definition 4. *A P system with symport/antiport rules and membrane division of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$, where:*

1. *$\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$ is a P system with symport/antiport rules.*
2. *$\mathcal{R}_1, \cdots, \mathcal{R}_q$ are finite set of symport/antiport rules which can also contain rules of the following form: $[a]_i \to [b]_i[c]_i$, where $i \notin \{1, i_{out}\}$ and $a, b, c \in \Gamma$* (division rules).

A division rule $[a]_i \to [b]_i[c]_i \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if the following holds: (a) membrane i is in $\mathcal{C}_t$; (b) object a is contained in such membrane; and (c) membrane i is neither the skin membrane nor the output membrane (if $i_{out} \in \{1, \ldots, q\}$). When applying a division rule $[a]_i \to [b]_i[c]_i$, under the influence of object a, the membrane with label i is divided into two membranes with the same label; in the first copy, object a is replaced by object b, in the second one, object a is replaced by object c; all the other objects residing in membrane i are replicated and copies of them are placed in the two new membranes.

Definition 5. *A P system with symport/antiport rules and membrane separation of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, \Gamma_0, \Gamma_1, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$$

where:

1. *$\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{out})$ is a P system with symport/antiport rules.*
2. *$\{\Gamma_0, \Gamma_1\}$ is a partition of Γ, that is, $\Gamma = \Gamma_0 \cup \Gamma_1$, $\Gamma_0, \Gamma_1 \neq \emptyset$, $\Gamma_0 \cap \Gamma_1 = \emptyset$;*
3. *$\mathcal{R}_1, \cdots, \mathcal{R}_q$ are finite set of rules symport/antiport rules which can also contain rules of the following form: $[a]_i \to [\Gamma_0]_i[\Gamma_1]_i$, where $i \notin \{1, i_{out}\}$ and $a \in \Gamma$* (separation rules).

A separation rule $[a]_i \to [\Gamma_0]_i[\Gamma_1]_i \in \mathcal{R}_i$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if the following holds: (a) membrane i is in $\mathcal{C}_t$; (b) object a is contained in such membrane; and (c) membrane i is neither the skin membrane nor the output membrane (if $i_{out} \in \{1, \ldots, q\}$). When applying a separation rule $[a]_i \to [\Gamma_0]_i[\Gamma_1]_i \in \mathcal{R}_i$, in reaction with an object a, the membrane i is separated into two membranes with the same label; at the same time, object a is consumed; the objects from Γ_0 are placed in the first membrane, those from Γ_1 are placed in the second membrane.

With respect to the semantics of these variants, the rules of such P systems are applied in a non-deterministic maximally parallel manner (at each step we apply a multiset of rules which is maximal, no further applicable rule can be added), with the following important remark: when a membrane i is divided (resp. separated), the division rule (resp. separation rule) is the only one from $\mathcal{R}_i$ which is applied for that membrane at that step (however, some rules can be applied in a daughter membrane). The new membranes resulting from division (resp. separation) could

participate in the interaction with other membranes or the environment by means of communication rules at the next step – providing that they are not divided (resp. separated) once again. The label of a membrane precisely identify the rules which can be applied to it.

The concept of recognizer is extended to P systems with symport/antiport rules and membrane division or membrane separation, in a natural way. We denote by $\mathbf{CDC}(k)$ (resp. $\mathbf{CSC}(k)$) the class of recognizer P systems with symport/antiport rules and membrane division (resp. membrane separation) such that the communication rules of the system have length at most k.

4 Non Efficiency of P Systems from CDC(1)

In this section, we study the limitations of efficient computations in systems from $\mathbf{CDC}(1)$. Specifically, we show that $\mathbf{P} = \mathbf{PMC}_{\mathbf{CDC(1)}}$, that is, the polynomial complexity class associated with the class of recognizer P systems $\mathbf{CDC}(1)$ is equal to the class $\mathbf{P}$.

Let $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$ be a recognizer P system from $\mathbf{CDC}(1)$. We denote by $\mathcal{M}_j^*$ the multiset over $\Gamma \times \{j\}$ obtained from $\mathcal{M}_j$ by replacing $a \in \Gamma$ by (a, j), and for each finite multiset w over Σ, we denote w^* the multiset over $\Sigma \times \{i_{in}\}$ obtained from $\mathcal{M}_j$ by replacing $a \in \Sigma$ by (a, i_{in})

The rules from $\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_q$ are of the following form: (a, out), (b, in) and $[a]_i \to [b]_i\ [c]_i$. These rules can be considered, in a certain sense, as a *dependency* between the object triggering the rule and the object produced by its application.

- The rules in $\mathcal{R}_i$ of type (a, out) can be described as the pair (a, i) produces the pair $(a, p(i))$.
- The rules in $\mathcal{R}_i$ of type (b, in) can be described as the pair $(b, p(i))$ produces the pair (b, i).
- The rules in $\mathcal{R}_i$ of type $[a]_i \to [b]_i\ [c]_i$ can be described as the pair (a, i) produces the pairs (b, i) and (c, i).

We formalize these ideas in the following definition.

Definition 6. *Let $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$ be a recognizer P system from $\mathbf{CDC}(1)$. The dependency graph associated with Π is the directed graph $G_\Pi = (V_\Pi, E_\Pi)$ defined as follows:*

- *The set of vertices is $V_\Pi = \{s\} \cup VL_\Pi \cup VR_\Pi$, where:*
  $VL_\Pi = \{(a, i) \in \Gamma \times \{0, \ldots, q\} \mid [(a, out) \in \mathcal{R}_i] \vee [\exists j \in ch(i)((a, in) \in \mathcal{R}_j)] \vee$
  $[\exists b, c \in \Gamma\ ([a]_i \to [b]_i[c]_i \in \mathcal{R}_i])\}$
  $VR_\Pi = \{(a, i) \in \Gamma \times \{0, \ldots, q\} \mid [(a, in) \in \mathcal{R}_i] \vee [\exists j \in ch(i)((a, out) \in \mathcal{R}_j)] \vee$
  $[\exists b, c \in \Gamma([b]_i \to [a]_i[c]_i \in \mathcal{R}_i)]\}$.
- *The set of edges is:*
  $E_\Pi = \{(s, (a, j)) \mid 1 \leq j \leq q \wedge (a, j) \in \mathcal{M}_j^*\} \cup$
  $\{((a, i), (b, j)) \in V_\Pi \times V_\Pi \mid [a = b] \wedge [j = p(i) \wedge (a, out) \in \mathcal{R}_i] \vee$

$$[a = b] \wedge [i = p(j) \wedge (a, in) \in \mathcal{R}_j] \vee$$
$$[i = j] \wedge [\exists c \in \Gamma\ ([a]_i \rightarrow [b]_i[c]_i \in \mathcal{R}_i)]\}.$$

In what follows, we show that the dependency graph associated with a P system from **CDC**(1), can be constructed by a single deterministic Turing machine working in polynomial time.

Proposition 1. *Let $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$ be a recognizer P systems from* **CDC**(1)*. There exists a Turing machine that constructs the dependency graph, G_Π, associated with Π, in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of rules and the maximum length of the rules).*

Proof. A deterministic algorithm that, given a recognizer P system Π from **CDC**(1), whose set of rules is $\mathcal{R} = \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_q$, constructs the corresponding dependency graph, is the following:

```
Input: (Π, R)
V_Π ← {s}; E_Π ← ∅
for j = 1 to q do
  for each pair (a, j) ∈ M_j^* do
   E_Π ← E_Π ∪ {(s, (a, j))}
  end for
end for
for each rule r ∈ R of Π do
  if r = (a, in) ∈ R_i then
    V_Π ← V_Π ∪ {(a, p(i)), (a, i)}; E_Π ← E_Π ∪ {((a, p(i)), (a, i))}
  end if
  if r = (a, out) ∈ R_i then
    V_Π ← V_Π ∪ {(a, i), (a, p(i))}; E_Π ← E_Π ∪ {((a, i), (a, p(i)))}
  end if
  if r = [a]_i → [b]_i[c]_i ∈ R_i then
    V_Π ← V_Π ∪ {(a, i), (b, i), (c, i)};
    E_Π ← E_Π ∪ {((a, i)), (b, i))} ∪ {((a, i), (c, i))}
  end if
end for
```

The running time of this algorithm is bounded by $O(|\mathcal{R}|) \subset O(q \cdot |\Gamma|^3)$. □

Proposition 2. *Let $\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$ be a recognizer confluent P system from* **CDC**(1)*. The following assertions are equivalent:*

(1) There exists an accepting computation of Π.
(2) There exists a path (with length greater or equal than 2) from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π.

Proof. (1) $\Rightarrow$ (2). First, we show that for each accepting computation $\mathcal{C}$ of Π there exists a path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π. By induction on the length n of $\mathcal{C}$.

Let $n = 1$ and $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1)$ be an accepting computation of Π with length 1. Then, a rule of the form $(\mathtt{yes}, out) \in \mathcal{R}_1$, with $a \in \Gamma$, has been applied at initial configuration $\mathcal{C}_0$. Then, $\mathtt{yes} \in \mathcal{C}_0(1)$, so $(\mathtt{yes}, 1) \in \mathcal{M}_1^*$. Hence, $(s, (\mathtt{yes}, 1), (\mathtt{yes}, 0))$ is a path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π.

Let us suppose that the result holds for n. Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n, \mathcal{C}_{n+1})$ be an accepting computation of Π with length $n + 1$. In this situation, $\mathcal{C}' = (C_1, \ldots, C_n, C_{n+1})$ is an accepting computation of the system $\Pi' = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}'_1, \ldots, \mathcal{M}'_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out})$, being $\mathcal{M}'_j = \{(a, i) \in \Gamma \times \{0, \ldots, q\} \mid C_1(j) = a\}$ the "content" of membrane j in configuration $\mathcal{C}_1$, for $1 \leq j \leq q$. By induction hypothesis there exists a path $\gamma_{\mathcal{C}'} = (s, (b_1, i_1), \ldots, (\mathtt{yes}, 0))$ from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π' (with length greater or equal than 2). We distinguish two cases. If $b_1 \in \mathcal{C}_0(i_1)$ (that means that in the first step of computation $\mathcal{C}$, a division rule has been applied to membrane i_1 such that object b_1 does not appear in the rule), then $\gamma_{\mathcal{C}} = (s, (b_1, i_1), \ldots, (\mathtt{yes}, 0))$ is a path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π, and the result holds. Otherwise, there is an element $b_0 \in \mathcal{C}_0(i_0)$ producing (b_1, i_1) at the first step of computation $\mathcal{C}$. Hence, $\gamma_{\mathcal{C}} = (s, (b_0, i_0), (b_1, i_1), \ldots, (\mathtt{yes}, 0))$ is a path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π.

(2) $\Rightarrow$ (1). Let us see that for each path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with Π, with length $k \geq 2$, there exists an accepting computation of Π. By induction on the length k of the path.

Let $k = 2$ and $(s, (a_0, i_0), (\mathtt{yes}, 0))$. Then, $i_0 = 1$ is the label of the skin membrane, $(a_0, out) \in \mathcal{R}_1$, $a_0 = \mathtt{yes}$, and the computation $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1)$ where the rule $(a_0, out) \in \mathcal{R}_1$ belongs to the multiset of rules that yields configuration C_1 from C_0, is an accepting computation of Π.

Let us suppose that the result holds for $k \geq 2$. Let

$$(s, (a_0, i_0), (a_1, i_1), \ldots (a_{k-1}, i_{k-1}), (\mathtt{yes}, 0))$$

be a path from s to $(\mathtt{yes}, 0)$ in the dependency graph of length $k + 1$. If $(a_0, i_0) = (a_1, i_1)$, then the result holds by induction hypothesis. Otherwise, let $\mathcal{C}_1$ be a configuration of Π reached from $\mathcal{C}_0$ by the application of a multiset of rules containing a rule that yields (a_1, i_1) from (a_0, i_0). Then $(s, (a_1, i_1), \ldots (a_{k-1}, i_{k-1}), (\mathtt{yes}, 0))$ is a path from s to $(\mathtt{yes}, 0)$ of length k, in the dependency graph of associated with the system

$$\Pi' = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}'_1, \ldots, \mathcal{M}'_q, \mathcal{R}_1, \cdots, \mathcal{R}_q, i_{in}, i_{out}),$$

where $\mathcal{M}'_j = \{(a, i) \mid \mathcal{C}_1(j)\}$ is the content of membrane j in configuration $\mathcal{C}_1$, for $1 \leq j \leq q$. By induction hypothesis, there exists an accepting computation $\mathcal{C}' = (\mathcal{C}_1, \ldots, \mathcal{C}_t)$ of Π'. Hence, $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_t)$ is an accepting computation of Π. □

Corollary 1. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ be a family of recognizer P systems from* $\mathbf{CDC}(1)$ *solving X, according to Definition 3. Let (cod, s) be the polynomial encoding associated with that solution. Then, for each instance w of the problem X the following assertions are equivalent:*

(a) $\theta_X(w) = 1$ *(that is, the answer to the problem is* `yes` *for w).*
(b) There exists a path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with the system $\Pi(s(w))$ with input multiset $cod(w)$.

Proof. Let $w \in I_X$. Then $\theta_X(w) = 1$ if and only if there exists an accepting computation of the system $\Pi(s(w)) + cod(w)$. Bearing in mind that $\Pi(s(w)) + cod(w)$ is a confluent system, from Proposition 4 we deduce that $\theta_X(w) = 1$ if and only if there exists a path from s to $(\mathtt{yes}, 0)$ in the dependency graph associated with the system $\Pi(s(w)) + cod(w)$. □

Theorem 1. $\mathbf{P} = \mathbf{PMC}_{\mathbf{CDC(1)}}$

Proof. We have $\mathbf{P} \subseteq \mathbf{PMC}_{\mathbf{CDC(1)}}$ because $\mathbf{PMC}_{\mathbf{CDC(1)}}$ is a nonempty class closed under polynomial–time reduction. Next, we show that $\mathbf{PMC}_{\mathbf{CDC(1)}} \subseteq \mathbf{P}$. For that, let $X \in \mathbf{PMC}_{\mathbf{CDC(1)}}$ and let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems from $\mathbf{CDC}(1)$ solving X, according to Definition 3. Let (cod, s) be the polynomial encoding associated with that solution.

We consider the following deterministic algorithm:

`Input: An instance` w `of` X
- `Construct the system` $\Pi(s(w)) + cod(w)$.
- `Construct the dependency graph` $G_{\Pi(s(w))+cod(w)}$ `associated with` $\Pi(s(w)) + cod(w)$.
- `Reachability` $(G_{\Pi(s(w))+cod(w)}, s, (\mathtt{yes}, 0))$

Obviously this algorithm is polynomial in the size $|w|$ of the input. □

5 Non efficiency of P systems from CSC(1)

In this section, we show that the polynomial complexity class associated with the class of recognizer P systems with symport/antiport rules and membrane separation is equal to the class $\mathbf{P}$, when we consider only communication rules with length 1.

In order to associate a dependency graph with each P system from $\mathbf{CSC}(1)$, let us notice that the application of a membrane separation rule $[a]_i \to [\Gamma_0]_i\ [\Gamma_1]_i$ consumes object a and the remaining objects in that membrane are separated in two membranes with the same label.

Definition 7. *Let Π be a recognizer P system from* $\mathbf{CSC}(1)$ *whose set of rules is* $\mathcal{R} = \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_q$. *The dependency graph associated with* Π *is the directed graph* $G_\Pi = (V_\Pi, E_\Pi)$ *defined as follows:*

- *The set of vertices is* $V_\Pi = \{s\} \cup VL_\Pi \cup VR_\Pi$, *where:*
  $VL_\Pi = \{(a,i) \in \Gamma \times \{0,\ldots,q\} : [(a,out) \in \mathcal{R}_i] \vee [\exists j \in ch(i)\ ((a,in) \in \mathcal{R}_j)] \vee [[a]_i \rightarrow [\Gamma_0]_i[\Gamma_1]_i \in \mathcal{R}_i])\}$
  $VR_\Pi = \{(a,i) \in \Gamma \times \{0,\ldots,q\} : [(a,in) \in \mathcal{R}_i] \vee [\exists j \in ch(i)((a,out) \in \mathcal{R}_j)]\}$.
- *The set of edges is*
  $E_\Pi = \{(s,(a,j)) \mid 1 \leq j \leq q \wedge (a,j) \in \mathcal{M}_j^*\} \cup \{((a,i),(a,j)) \in V_\Pi \times V_\Pi : [j = p(i) \wedge (a,out) \in \mathcal{R}_i] \vee [i = p(j) \wedge (a,in) \in \mathcal{R}_j]\}$.

In a similar way as in the previous section, the following results are obtained.

Proposition 3. *Let Π be a recognizer P system from* $\mathbf{CSC}(1)$. *There exists a Turing machine that constructs the dependency graph G_Π associated with Π, in polynomial time.*

Proposition 4. *Let Π be a recognizer confluent P system from* $\mathbf{CSC}(1)$. *The following assertions are equivalent:*

(1) There exists an accepting computation of Π.
(2) There exists a path (with length greater or equal than 2) from s to $(\mathtt{yes}, 0)$ *in the dependency graph associated with Π.*

Theorem 2. $\mathbf{P} = \mathbf{PMC}_{\mathbf{CSC(1)}}$

6 Conclusions and Further Works

In the framework of (cell-like) P systems with symport/antiport rules, two new kind of rules inspired by the processes of *mitosis* and *membrane fission* in eukaryotic cells, have been considered, called P systems with symport/antiport rules and membrane division or membrane separation.

By using the *dependency graph* technique, the computational efficiency of these P systems has been studied in the case of non-cooperative systems, that is, systems with communication rules of length 1.

For future work, we plan to establish the efficiency of these kind of P systems in order to obtain borderline of the efficiency of the problems in terms of syntactical ingredients of P systems with symport/antiport rules.

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (61033003, 91130034 and 61320106005). The work of M.J. Pérez-Jiménez and L.F. Macías-Ramos was supported by Project TIN2012-37434 of the Ministerio de Ciencia e Innovación of Spain.

References

1. T.H. Cormen, C.E. Leiserson, R.L. Rivest. *An Introduction to Algorithms*. The MIT Press, Cambridge, Massachussets, 1994.
2. P. Frisco. *Computing with Cells: Advances in Membrane Computing*, Oxford University Press, Oxford, 2009.
3. R. Gutiérrez-Escudero, M.J. Pérez-Jiménez, M. Rius-Font. Characterizing tractability by tissue-like P systems. *Lecture Notes in Computer Science* **5957**, 5957 (2010), 289-300.
4. M. Ionescu, Gh. Păun, T. Yokomori. Spiking neural P systems, *Fundamenta Informaticae*, **71**, 2-3 (2006), 279–308.
5. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodriguez-Paton. Tissue P systems, *Theoretical Computer Science*, **296**, 2 (2003), 295–326.
6. L. Pan, T.-O. Ishdorj. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5 (2004), 630–649.
7. L. Pan, M.J. Pérez-Jiménez. Computational complexity of tissue–like P systems. *Journal of Complexity*, **26**, 3 (2010), 296–315.
8. A. Păun, Gh. Păun, G. Rozenberg. Computing by communication in networks of membranes, *International Journal of Foundations of Computer Science*, **13**, 6 (2002), 779–798
9. A. Păun, Gh. Păun. The power of communication: P systems with symport/antiport, *New Generation Computing*, **20**, 3 (2002), 295–305.
10. Gh. Păun. Computing with membranes, *Journal of Computer and Systems Science*, **61**, 1 (2000), 108–143.
11. Gh. Păun. P systems with active membranes: attacking **NP**–complete problems, *Journal of Automata, Languages and Combinatorics*, **6** (2001), 75–90.
12. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez. Tissue P system with cell division. *International Journal of Computers, Communications & Control* , **Vol. III**, 3 (2008), 295–303.
13. Gh. Păun, G. Rozenberg, A. Salomaa (eds.). *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford, 2010.
14. Gh. Păun. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.
15. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, F. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
16. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, A polynomial complexity class in P systems using membrane division, *Journal of Automata, Languages and Combinatorics*, **11**, 4 (2006) 423–434.

Sparse-matrix Representation of Spiking Neural P Systems for GPUs

Miguel Á. Martínez-del-Amor[1], David Orellana-Martín[1], Francis G.C. Cabarle[2], Mario J. Pérez-Jiménez[1], Henry N. Adorna[2]

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

Summary. Current parallel simulation algorithms for Spiking Neural P (SNP) systems are based on a matrix representation. This helps to harness the inherent parallelism in algebraic operations, such as vector-matrix multiplication. Although it has been convenient for the first parallel simulators running on Graphics Processing Units (GPUs), such as CuSNP, there are some bottlenecks to cope with. For example, matrix representation of SNP systems with a low-connectivity-degree graph lead to sparse matrices, i.e. containing more zeros than *actual* values. Having to deal with sparse matrices downgrades the performance of the simulators because of wasting memory and time.

However, sparse matrices is a known problem on parallel computing with GPUs, and several solutions and algorithms are available in the literature. In this paper, we briefly analyse some of these ideas and apply them to represent some variants of SNP systems. We also conclude which variant better suit a sparse-matrix representation.

Keywords: Spiking Neural P systems, Simulation Algorithm, Sparse Matrix Representation, GPU computing, CUDA

1 Introduction

Spiking Neural P (SNP) systems [9] are a type of P systems [16] composed of a directed graph inspired by how neurons are interconnected by axons and synapses

in the brain. Neurons communicate through spikes, and the time difference between them plays an important role in the computation.

The simulation of SNP systems have been carried out through sequential simulators such as pLinguaCore [11]. For parallel simulation, a matrix representation was introduced [17], so that the simulation algorithm is based on applying matrix operations. For instance, efficient algebra libraries have been defined for GPUs, given that they fit well to the highly parallel architecture of these devices. This have been harnessed already to introduce the first parallel SNP system simulators on GPUs, cuSNP [4, 5].

However, this matrix representation can be sparse, having a majority of zero values, because the directed graph of SNP systems are not normally fully connected. In many disciplines, sparse vector-matrix operations are natural, so many solutions have been proposed in the literature [6]. For this reason, we transfer some of these ideas to the simulation of SNP systems with matrix operations. First, we give a first approach, which is further optimized by splitting the main matrix into several structures. Second, ideas to deal with dynamic networks are given.

The paper is structured as follows: Section 2 gives a short formal definition of SNP systems; Section 3 summarizes the matrix-based simulation algorithm; Section 4 briefly introduces GPU computing and sparse vector-matrix representations; Section 5 discussed the ideas on introducing sparse vector-matrix representation for SNP system simulation; and the paper finishes with conclusions and future work.

2 Spiking Neural P Systems

Definition 1. *A* spiking neural P system *of degree* $q \geq 1$ *is a tuple*

$$\Pi = (O, syn, \sigma_1, \ldots, \sigma_q, i_{out})$$

where:

- $O = \{a\}$ *is the singleton alphabet (a is called spike);*
- $syn = (V, E)$ *is a directed graph such that* $V = \{\sigma_1, \ldots, \sigma_q\}$ *and* $(\sigma_i, \sigma_i) \notin E$ *for* $1 \leq i \leq q$*;*
- $\sigma_1, \ldots, \sigma_m$ *are neurons of the form*

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

  where:
  - $n_i \geq 0$ *is the initial number of spikes within neuron labelled by* i*; and*
  - R_i *is a finite set of rules associated to neuron labelled by* i*, of the following forms:*
    (1) $E/a^c \rightarrow a^p$*, being* E *a regular expression over* $\{a\}$*,* $c \geq p \geq 1$ *(firing rules);*

(2) $a^s \to \lambda$ *for some* $s \geq 1$, *with the restriction that for each rule* $E/a^c \to a^p$ *of type of type* (1) *from* R_i, *we have* $a^s \notin L(E)$ *(forgetting rules).*

- $i_{out} \in \{1, 2, \ldots, q\}$ *such that* $outdegree(i_{out}) = 0$

A spiking neural P system of degree $q \geq 1$ can be viewed as a set of q neurons $\{\sigma_1, \ldots, \sigma_q\}$ interconnected by the arcs of a directed graph *syn*, called *synapse graph* There is a distinguished neuron i_{out}, called output neuron, which communicates with the environment.

If a neuron σ_i contains k spikes at an instant t, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a^p$ can be applied. By the application of that rule, c spikes are removed from neuron σ_i and the neuron fires producing p spikes immediately. The spikes produced by a neuron σ_i are received for all neuron σ_j such that $(\sigma_i, \sigma_j) \in E$. If σ_i is the output neuron then the spikes are sent to the environment.

The rules of type (2) are *forgetting* rules, and they are applied as follows: If neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be applied. By the application of this rule all s spikes are removed from σ_i.

In spiking neural P systems, a global clock is assumed, marking the time for the whole system. Only one rule can be executed in each cell at step t. As models of computation, spiking neural P systems are *Turing complete*, i.e. as powerful as Turing machines. On one hand, common way to introduce input to the system is to encode into some or all of the n_i's the input(s) of the problem to be solved. On the other hand, a common way to obtain the output is by observing i_{out}: either by getting the interval $t_2 - t_1 = n$, where i_{out} sent its first two spikes at times t_1 and t_2 (we say n is computed or generated by the system), or by counting all the spikes sent by i_{out} to the environment until the system halts.

Aside from computing numbers, spiking neural P systems can also compute strings, and hence, languages. More general ways to provide the input or receive the output include the use of *spike trains*, i.e. a stream or sequence of spikes entering or leaving the system. Further results and details on computability, complexity, and applications of spiking neural P systems are detailed in [15], a dedicated chapter in the Handbook in [8], and an extensive bibliography until February 2016 in [14]. There are some interesting ingredients we are going to explain here. A broader explanation of them and more variants is provided at [1, 3, 13].

2.1 Spiking Neural P Systems with Budding

Based on the idea of neuronal budding, where a cell is divided in two new cells, we can abstract it to *budding rules*. In this process, the new cells can differ in some aspects: their connections, contents and shape. A budding rule has the following form:

$$[E]_i \to [\ ]_i / [\ ]_j,$$

where E is a regular expression and $i, j \in \{1, \ldots, q\}$.

If a neuron σ_i contains s spikes, $a^s \in L(E)$, and there is no neuron σ_j such that there exists a synapse (i, j) in the system, then the rule $[E]_i \to [\ ]_i / [\ ]_j$ is enabled

and it can be executed. A new neuron σ_j is created, and both neurons σ_i and σ_j are empty after the execution of the rule. This neuron σ_i keeps all the synapses that were going in, and this σ_j inherits all the synapses that were going out of σ_i in the previous configuration. There is also a synapse (i, j) between neurons σ_i and σ_j, and the rest of synapses of σ_j are given to the neuron depending on the synapses of *syn*.

2.2 Spiking Neural P Systems with Division

Inspired by the process of *mitosis*, division rules have been widely used within the field of *Membrane Computing*. In SN P systems, a division rule can be defined as follows:

$$[E]_i \rightarrow [\ ]_j | | [\ ]_k,$$

where E is a regular expression and $i, j, k \in \{1, \ldots, q\}$.

If a neuron σ_i contains s spikes, $a^s \in L(E)$, and there is no neuron σ_g such that the synapse (g, i) or (i, g) exists in the system, $g \in \{j, k\}$, then the rule $[E]_i \rightarrow [\ ]_j | | [\ ]_k$ is enabled and it can be executed. Neuron σ_i is then divided into two new cells, σ_j and σ_k. The new cells are empty at the time of their creation. The new neurons keep the synapses previously associated to the original neuron σ_i, that is, if there was a synapse from σ_g to σ_i, then a new synapse from σ_g to σ_j and a new one to σ_k are created, and if there was a synapse from σ_i to σ_g, then a new synapse from σ_j to σ_g and a new one from σ_k to σ_g are created. The rest of synapses of σ_j and σ_k are given by the ones defined in *syn*.

2.3 Spiking Neural P Systems with Plasticity

It is known that new synapses can be created in the brain thanks to the process of *synaptogenesis*. We can recreate this process in the framework of spiking neural P systems defining *plasticity rules* in the following form:

$$E/a^c \rightarrow \alpha k(i, N_j),$$

where E is a regular expression, $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $k \geq 1$ and $N_j \subseteq \{1, \ldots, q\}$. For a neuron σ_i, let us define the *presynapses* of this neuron as $pres(i) = \{j | (i, j) \in syn\}$.

If a neuron σ_i contains s spikes, $a^s \in L(E)$, then the rule $E/a^c \rightarrow \alpha k(i, N_j)$ is enabled and can be executed. The rule consumes c spikes and, depending on the value of α, it performs one of the following:

- If $\alpha = +$ and $N_j - pres(i) = \emptyset$, or if $\alpha = -$ and $pres(i) = \emptyset$, then there is nothing more to do.
- If $\alpha = +$ and $|N_j - pres(i)| \leq k$, deterministically create a synapse to every σ_g, $g \in N_j - pres(i)$. Otherwise, if $|N_j - pres(i)| > k$, then non-deterministically select k neurons in $N_j - pres(i)$ and create one synapse to each selected neuron.

- If $\alpha = -$ and $|pres(i)| \leq k$, deterministically delete all synapses in $pres(i)$. Otherwise, if $|pres(i)| > k$, then non-deterministically select k neurons in $pres(i)$ and delete each synapse to the selected neurons.
- If $\alpha = \{\pm, \mp\}$, create (respectively, delete) synapses at time t and then delete (resp., create) synapses at time $t + 1$. Even when this rule is applied, neurons are still open, that is, they can continue receiving spikes.

Let us notice that if, for some σ_i, we apply a *plasticity rule* with $\alpha \in \{+, \pm, \mp\}$, when a synapse is created, a spike is sent from σ_i to the neuron that has been connected. That is, when σ_i attaches to σ_j through this method, we have immediately transferring one spike to σ_j.

3 Simulation of SNP Systems

So far, P system parallel simulators make use of *ad-hoc* representations, specifically defined for a certain variant [12]. In order to ease the simulation of SNP system and its deployment to parallel environments, a matrix representation was introduced [17]. By using a set of algebraic operations, it is possible to reproduce the transitions of a computation. Although the baseline representation only involves SNP systems without delays and static structure, many extensions have followed such as for enabling delays or supporting non-determinism [4, 5].

This representation includes the following vectors and matrices, for a SNP system π of degree (n, m) (n rules and m neurons):

Configuration vector: C_k is the vector containing all spikes in every neuron on the k^{th} computation step/time, where C_0 denotes the initial configuration. It contains m elements.

Spiking vector: S_k shows if a rule is going to fire at the transition step k (having value *1*) or not (having value *0*). Given the non-determinism nature of SNP systems, it would be possible to have a set of valid spiking vectors. However, for the computation of the next configuration vector, only a spiking vector is used. It contains n elements.

Spiking transition matrix: M_π is a matrix comprised of a_{ij} elements where a_{ij} is given as

Definition 2.

$$a_{ij} = \begin{cases} -c, \textit{ rule } r_i \textit{ is in } \sigma_j \textit{ and is applied consuming } c \textit{ spikes;} \\ p, \textit{ rule } r_i \textit{ is in } \sigma_s \textit{ } (s \neq j \textit{ and } (s, j) \in syn) \\ \quad \textit{and is applied producing } p \textit{ spikes in total;} \\ 0, \textit{ rule } r_i \textit{ is in } \sigma_s \textit{ } (s \neq j \textit{ and } (s, j) \notin syn). \end{cases}$$

Thus, rows represent rules and columns represent neurons in the spiking transition matrix. Note also that a negative entry corresponds to the consumption of spikes. Thus, it is easy to observe that each row has exactly one negative entry, and each column has at least one negative entry [17]. It contains $n \cdot m$ elements.

Hence, to compute the transition k, it is enough to select a spiking vector S_k and calculate: $C_k = S_k \cdot M_\pi + C_{k-1}$.

4 GPU Computing and SpMV Operations

The Graphics Processing Unit (GPU) has been employed for P system simulations since the introduction of CUDA . This technology allows to run scientific computations in parallel on the GPU, given that a device typically contains thousands of cores and high memory bandwidth [10]. However, parallel computing on a GPU has more constraints than on a CPU: threads have to run in a SIMD fashion, accessing data in a coalesced way; that is, best performance is achieved when the execution of threads is synchronized and accessing contiguous data from memory.

Some algorithms fit perfectly to the GPU parallel model, such as algebraic operations. Indeed, matrix computation is a "hello world" when getting started with CUDA [18], and there are many efficient libraries for algebra computations like cuBLAS. It is usual that when working with large matrices, these are almost "empty", or with a majority of zero values. This is known as *sparse matrix*, and this downgrades the runtime in two ways: lot of memory is wasted, and lot of operations are redundant.

Given the importance of linear algebra in many computational disciplines, sparse vector-matrix operations (*SpMV*, in short) have been subject of study in parallel computing (and so, on GPUs). Today there exists many approaches to tackle this problem [2]. In this paper, we will focus on two formats to represent sparse matrices, assuming that threads will access rows in parallel:

- *CSR* format. Only non-null values are represented by using 3 arrays: row pointers, non-zero values and columns (see Figure 1 for an example). First, the row-pointers array is accessed, which contains a position per row of the original matrix. Each position says the index where the row start in the non-zero values and columns arrays. The non-zero values and the columns arrays can be seen as a single array of pairs, since every entry has to be accessed at the same time. Once a row is indexed, then a loop over the values in that row has to be performed, so that the corresponding column is found, and therefore, the value. If the column is not present, then the value is assumed to be zero, since this data structures contains all non-zero values. The main advantage is that it is a full-compressed format if $NumNonZeroValues \cdot 2 > NumZeroValues$, where $NumNonZeroValues$ and $NumZeroValues$ are the number of non-zero and zero values in the original matrix, respectively. However, the drawbacks is that the search of elements in the non-zero values and columns arrays is not coalesced when using parallelism per row. Moreover, since it is a full-compressed format, there is no room for modifying the values, such as introducing new non-zero values.

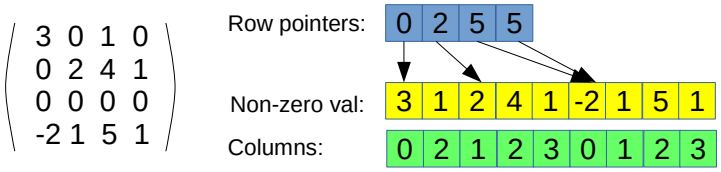


Fig. 1: CSR format example

- *ELL* format. This representation aims at increase the memory coalescing access of threads in CUDA. This is achieved by using a matrix of pairs, containing a trimmed, transposed version of the original matrix (see Figure 2 for an example). Each column of the ELL matrix is devoted for each row of the matrix, even though the row is empty (all elements are zero). Every element is a pair, where the first position denotes the column and the second is the value, of only the non-zero elements in the corresponding row. However, the size of the matrix is fixed, so the number of columns equals the number of rows of the original matrix, but the number of rows is the maximum length of a row in terms of non-zero values; in other words, the maximum amount of non-zero elements in a row of the original matrix. Rows containing fewer elements will pad the difference with null elements. The main advantage of this format is that threads will always access the elements of all rows in coalesced way, and the null elements padded by short rows can be utilize to incorporate new data. However, there is a waste of memory, which is worst when the rows are unbalance in terms of number of zeros.

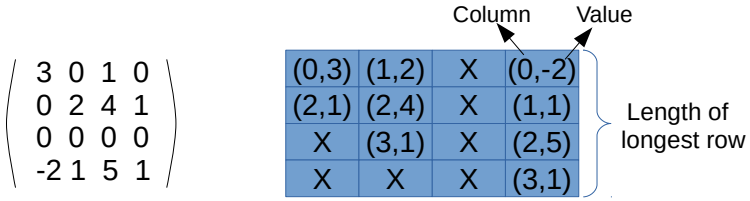


Fig. 2: ELL format example

5 Sparse Matrix Representation of SNP Systems

SNP systems in the literature typically are not fully connected graphs. In such situations, the transition matrix gets sparse, and therefore, further optimizations based on SpMV can be conveyed. In the following subsections, we discuss some

approaches. Of course, if the graph inherent to a SNP system leads to a dense transition matrix, then a normal format can be employed, because using sparse formats will increase the memory footprint.

5.1 Approach with ELL Format

The first approach to compress the representation of a sparse transition matrix, M_π, is to use the ELL format (see Figure 3 for an example), leading to the compressed matrix M_π^s. The following aspects have to be taken into consideration:

- The number of rows of M_π^s equals the maximum amount of non-zero values in a row of M_π, denoted by Z. It can be shown that $Z = MaxOutDegree + 1$, where $MaxOutDegree$ is the maximum output degree found in the neurons of the SNP system. Z can be derived from the composition of the transition matrix, where a row devoted for a rule $E/a^c \to a^p$ contains the values $+p$ for every neuron (columns) to which the neuron it belongs has a synapse, and a value $-c$ for consuming the spikes in the neuron it belongs.
- The values inside columns can be sorted, so that the consumption of spikes ($-c$ values) are placed at the first row. In this way, all threads can start with the same task, consuming spikes.
- Every position of M_π^s is a pair (although not represented in Figure 3), where the first element is a neuron label, and second is $+p$.

The idea is to assign a thread to each rule, and so, one per column of the spiking vector S_k and one per column of M_π^s (real rows of the transition matrix). For the vector-matrix multiplication, it is enough to iterate Z times (number of rows in M_π^s). In each iteration, the computed value is added to the corresponding neuron position in the configuration vector C_k. Since some threads will possibly write to the same positions in the configuration vector, a solution would be to use atomic operations, which are available on GPUs to calculate additions, among others.

5.2 Optimized Approach

If, in general, there are more than one rule in the neurons, lot of threads in the first approach will be inactive (having a 0 in the spiking vector), causing branch divergence and non-coalesced memory access. Moreover, note in Figure 3 that columns corresponding to rules belonging to the same neuron will contain redundant information: the generation of spikes is replicated for all synapses.

Therefore, a more efficient sparse matrix representation can be obtained when maintaining the synapses separated from the rule information. This can be done as follows:

- *Rule information.* By using a CSR-like format (see Figure 4 for an example), rules of the form $E/a^c \to a^p$ (also forgetting rules are included, assuming

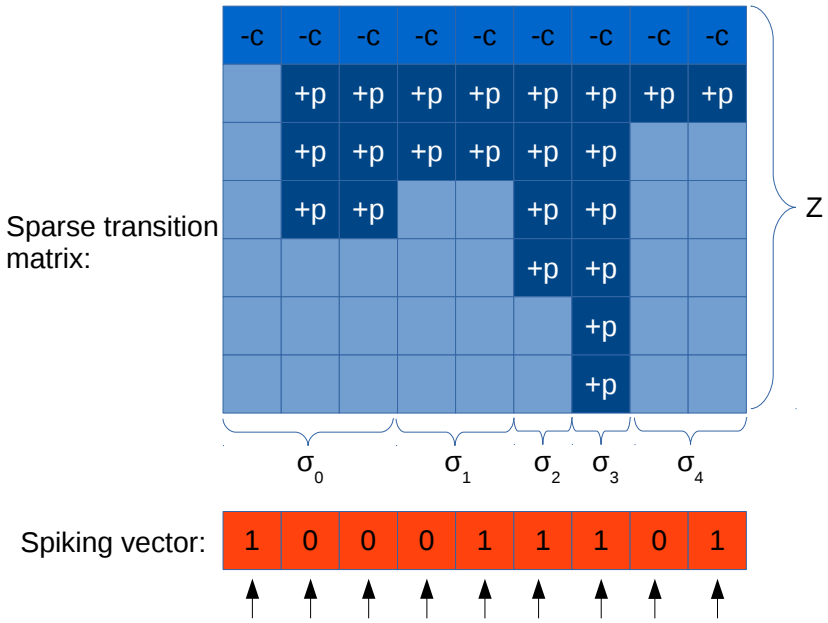


Fig. 3: Sparse matrix representation of a SNP system based on ELL format

$p = 0$) can be represented by a double array storing the values c and p (also the regular expression, but this is required only to select a spiking vector, and hence is out of scope of this work). A pointer array is employed to relate, for each neuron, the subset of rules that has associated.

- *Synapse matrix*, Sy_π. It has a column per neuron i, and a row for every neuron j such that $(i, j) \in Syn$ (there is a synapse). That is, every element of the matrix corresponds to a synapse or null, given that the number of rows equals to the maximum output degree in the neurons of the SNP system π, and padding is required.
- *Spiking vector* is modified, containing only m positions, one per neuron, and stating which rule $0 \leq r \leq n$ is selected.

The way to operate with this approach is to assign a thread to each column of the synapse matrix (requiring m threads, one per neuron). Each thread will access to the corresponding rule stated in the spiking vector, delete c in the configuration vector C_k, and add p to each neuron defined in synapse matrix in C_k.

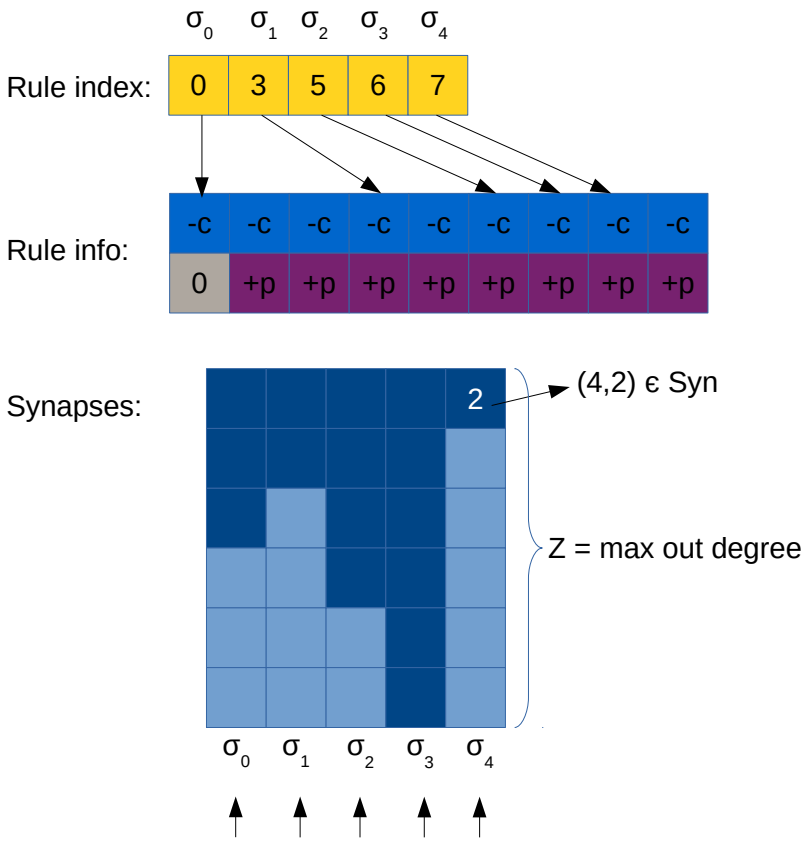


Fig. 4: Optimized sparse matrix representation

5.3 Ideas for Dynamic Networks

The optimized sparse matrix representation discussed in Section 5.2 can be further extended to support rules that modify the network, such as budding, division or plasticity.

Figure 5 shows an example on how a budding rule can be supported. First, the synapse matrix has to be flexible enough to host new neurons. This can be accomplished by allocating a matrix large enough to populate new neurons (probably up to fill the whole memory available on the GPU). Thus, for a budding rule $[E]_i \rightarrow []_j/[]_k$, the required operations to modify the synapse matrix are:

1. Allocate a column for the new neuron k.
2. Copy column i to k.

3. Delete content of column i and add only one element for k.
4. Change label i to j.

This will require to use a map of the column labels of the synapse matrix, saying to which neuron it corresponds. The same map can be used to index the rule information structure. A further optimization is to swap the labels i for k instead of copying the column content. One major drawback of this approach is that the creation of new neurons cannot run fully in parallel; that is, assigning new columns to created neurons in a transition step is a serialized process. Some techniques such as prefix sum can be applied to cope with this issue and convert a serial process into a logarithmic-step operation.

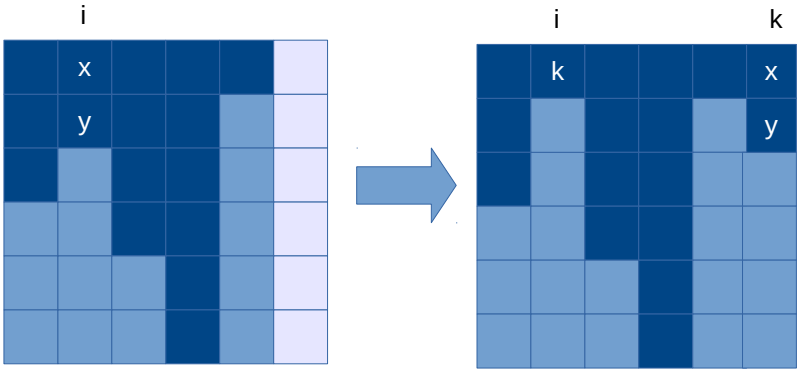


Fig. 5: Extension for budding

An example for division rules can be seen in Figure 6. Again, the synapse matrix has to be extended to contain empty columns to host populated new neurons during the simulation, and a map of neuron labels have to be managed.

For a division rule $[E]_i \rightarrow []_j || []_k$, the following operations have to be performed:

1. Allocate a new column for neuron k.
2. Copy column i to k.
3. Change label of i to j.
4. Find all occurrences of i in the synapse matrix, change it for j and add k in the column.

The last operation can be shown to be very expensive, since it requires to loop all over the synapse matrix. Moreover, when adding k in all the neurons containing i in the synapses, it would be possible to exceed the predetermined size Z. For this situation, a special array of overflows will be needed, like ELL+COO format for SpMV [2].

Finally, Figure 7 shows an example for plasticity rules. In this case, the synapse matrix can be allocated in advance to an exact size, since no new neurons

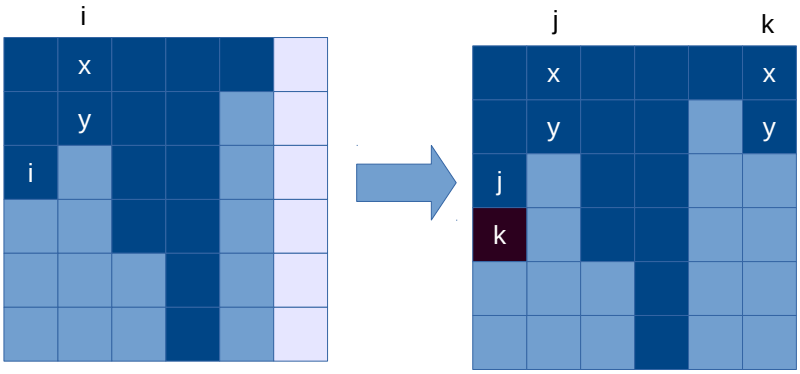


Fig. 6: Extension for division

are created. However, enough rows Z have to be pre-established to support the maximum amount of synapses, what can be precomputed by looking to the initial out degrees of the neurons and the size of the N sets in the plasticity rules for adding synapses: $E/a^c \rightarrow \alpha k(i, N_j)$, with $\alpha = +/\pm/\mp$.

The following operations have to be performed to reproduce the behaviour:

1. When deleting synapses, loop Z times to find the corresponding neurons in the matrix, and set them to null. Holes might appear in the columns.
2. When adding synapses, loop Z times to find holes in the column and add the corresponding neurons.

Since holes might appear in the columns when deleting synapses, we will need to loop over Z times every column to compute the next transition, or to add new synapses. Sorting algorithms can be run in parallel, but most probably it will not worth the effort.

6 Conclusions and Future Work

In this paper, we have analysed the problem of having sparse matrices in the matrix representation of SNP systems. Downgrades in the simulator performance would appear if no solutions are found. However, this is a known issue in other disciplines, and efficient sparse matrix representations have been introduced in the literature.

We proposed a two efficient sparse representations for SNP systems, one based on the classic format ELL, and an optimized one based on CSR and ELL. We also analysed their behaviour when supporting rules for dynamic networks: division, budding and plasticity. The representation for plasticity poses more advantages than the one for division and budding, since the synapse matrix size can be pre-computed. Thus, no label mapping nor empty columns for new neurons are

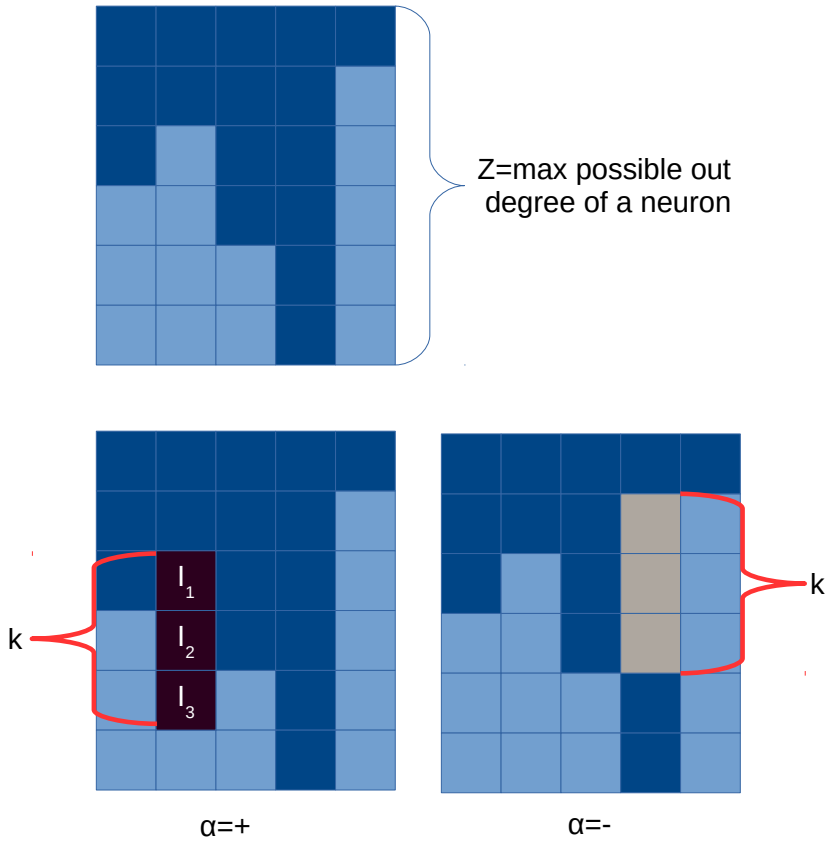


Fig. 7: Extension for plasticity

required. Moreover, simulating the creation of new neurons in parallel can damage the performance of the simulator significantly, because this operation can be sequential. Plasticity rules do not create new neurons, so this is avoided.

As future work, we plan to provide implementations of this ideas within cuSNP framework, and deep analyse the different results with real examples from the literature. We believe that this ideas will help to bring efficient tools to simulate SNP systems on GPUs, enabling the simulation of large networks in parallel.

References

1. H. Adorna, F. Cabarle, L.F. Macías-Ramos, L. Pan, M.J. Pérez-Jiménez, B. Song, T. Song, L. Valencia-Cabrera. Taking the pulse of SN P systems: A quick survey,

in: M. Gheorghe, I. Petre, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (eds.), *Multidisciplinary creativity*, Spandugino, 2015, pp. 3–16.
2. N. Bell, M. Garland. Efficient Sparse Matrix-Vector Multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, 2008.
3. F. Cabarle, H. Adorna, M.J. Pérez-Jiménez, T. Song. Spiking neural P systems with structural plasticity, *Neural Computing and Applications*, **26**, 8 (2015), pp. 1905–1917
4. J.P. Carandang, J.M.B. Villaflores, F.G.C. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor. CuSNP: Spiking Neural P Systems Simulators in CUDA. Romanian Journal of Information Science and Technology, **20**, 1 (2017), 57–70.
5. J.P. Carandang, F.G.C. Cabarle, H.N. Adorna, N.H.S. Hernandez, M.A. Martínez-del-Amor. Nondeterminism in Spiking Neural P Systems: Algorithms and Simulations. *Asian Conference on Membrane Computing* 2017. Submitted.
6. K. Fatahalian, J. Sugerman, P. Hanrahan. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication, *In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (HWWS '04)* , ACM, 2004, pp. 133-137.
7. M. Harris. Mapping computational concepts to GPUs, *ACM SIGGRAPH 2005 Courses*, NY, USA, 2005.
8. O. Ibarra, A. Leporati, A. Păun, S. Woodworth. Spiking Neural P Systems, in: Gh. Păun and G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010, pp. 337–362.
9. M. Ionescu, Gh. Păun, T. Yokomori. Spiking Neural P Systems, *Journal Fundamenta Informaticae*, **71**, 2-3 (2006), 279-308.
10. D. Kirk, W. Hwu, *Programming Massively Parallel Processors: A Hands On Approach*, 1st ed. MA, USA: Morgan Kaufmann, 2010.
11. L.F. Macías, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M.J. Pérez-Jiménez, A. Riscos-Núñez. A P-Lingua based simulator for Spiking Neural P systems, *Lecture Notes in Computer Science*, **7184** (2012), 257–281.
12. M.A. Martínez-del-Amor, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez. Simulating P systems on GPU devices: a survey. Fundamenta Informaticae, **136**, 3 (2015), 269-284.
13. L. Pan, Gh. Păun, M.J. Pérez-Jiménez. Spiking neural P systems with neuron division and budding, *Science China Information Sciences*, **54**, 8 (2011), 1596–1607.
14. L. Pan, T. Wu, Z. Zhang. A Bibliography of Spiking Neural P Systems. *Bulletin of the International Membrane Computing Society*, June 2016, 63–78.
15. Gh. Păun, M.J. Pérez-Jiménez. Algorithmic Bioprocesses, *Spiking neural P systems. Recent results, research topics*, Springer, 2009, pp. 273–291.
16. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and TUCS Report No 208.
17. X. Zeng, H. Adorna, M. A. Martínez-del-Amor, L. Pan, M. J. Pérez-Jiménez. Matrix representation of spiking neural p systems. *Lecture Notes in Computer Science*, **6501** (2011), 377–391.
18. NVIDIA corporation,*NVIDIA CUDA C programming guide*, version 3.0, CA, USA: NVIDIA, 2010.

P Systems with Active Cells

David Orellana-Martín

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad of Sevilla
dorellana@us.es

Summary. P systems with active membranes is a widely studied framework within the field of Membrane Computing since the creation of the discipline. The abstraction of the structure and behavior of living cells is reflected in the tree-like hierarchy and the kinds of rules that can be used in these kinds of systems.

Resembling the organization and communication between cells within tissues that form organs, tissue-like P systems were defined as their abstractions, using symport/antiport rules, that is, moving and exchanging elements from one cell to another one. All the cells are located in an environment where there exist an arbitrary number of some elements.

Lately, symport/antiport rules have been used in the framework of cell-like membrane systems in order to study their computational power. Interesting results have been reached, since they act similarly to their counterparts in the framework of tissue P systems.

Here, the use of the former defined rules (that is, evolution, communication, dissolution and division/separation rules) is considered, but not working with a tree-like structure. Some remarks about choosing good semantics are given.

Key words: Membrane Computing, Active cells, Computational Complexity, **P** versus **NP** problem.

1 Introduction

Membrane Computing is a distributed parallel computing paradigm inspired by the way the living cells process chemical substances, energy and information. The processor units in the basic model are abstractions of biological membranes, selectively permeable barriers which give cells their outer boundaries (plasma membranes) and their inner compartments (organelles). They control the flow of information between cells and the movement of substances into and out of cells and they are also involved in the capture and release of energy. Biological membranes play an active part in the life of the cell. In fact, the passing of a chemical substance

through a biological membrane is often implemented by an interaction between the membrane itself and the protein channels present in it. During this interaction, the chemical substance and the membrane itself can be modified at least locally.

P systems with active membranes [7] include rules inspired on the behavior of the proteins inside the cells. Recalling, evolution rules are the abstraction of the mutation of the chemical compounds within singular organelles, communication rules give us the idea of the transport of the proteins through the membranes of the cells, dissolution rules remember the process of *apoptosis*, which makes the cell to "kill itself" (in this case, we take the inspiration and apply it to membranes). At last, division and separation rules are the rules that can create an exponential workspace in polynomial time. These are inspired by the asexual and sexual cell processes, that give birth to new cells.

All of those rules can be successfully applied in the framework of tissue-like P systems. Moreover, it would be a more natural way to describe the functioning of these rules at the cells that in the membranes. As an analogy to *P systems with active membranes*, we are going to call them *P systems with active cells*

The paper is organized as follows. Next section briefly introduces some preliminaries needed to make the work self-contained. Section 3 will be devoted to present both syntax and semantics of tissue-like P systems with active cells, letting Section 4 dedicated to present some results concerning the computational complexity classes reached by this kind of membrane systems. The paper ends with some open problems and concluding remarks.

2 Preliminaries

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

A multiset over an alphabet Γ is an ordered pair (Γ, f) where f is a mapping from Γ onto the set of natural numbers $\mathbb{N}$. The *support* of a multiset $m = (\Gamma, f)$ is defined as $supp(m) = \{x \in \Gamma | f(x) > 0\}$. A multiset is finite (respectively, empty) if its support is a finite (respectively, empty) set. We denote by $\emptyset$ the empty multiset. We denote by $M_f(\Gamma)$ the set of all finite multisets over Γ. The *cardinal* of a finite multiset m is defined as $\sum_{x \in \Gamma} m(x)$.

Let $m_1 = (\Gamma, f_1)$, $m_2 = (\Gamma, f_2)$ be multisets over Γ, then the union of m_1 and m_2, denoted by $m_1 + m_2$, is the multiset (Γ, g), where $g(x) = f_1(x) + f_2(x)$ for each $x \in \Gamma$. We say that m_1 is contained in m_2 and we denote it by $m_1 \subseteq m_2$, if $f_1(x) \leq f_2(x)$ for each $x \in \Gamma$. The relative complement of m_2 in m_1, denoted by $m_1 \setminus m_2$, is the multiset (Γ, g), where $g(x) = f_1(x) - f_2(x)$ if $f_1(x) \geq f_2(x)$, and $g(x) = 0$ otherwise.

A *rooted tree* is a connected, acyclic, undirected graph in which one of the vertices (called the *root of the tree*) is distinguished from the others. Given a node x (different from the root) in a rooted tree, if the last edge on the (unique) path from the root to the node x is $\{x, y\}$ (so $x \neq y$), then y is **the** *parent* of node x and x is **a** *child* of node y. We denote it by $y = p(x)$ and $x \in ch(y)$. The root is the only node in the tree with no parent. A node with no children is called a *leaf* (see [2] for details).

Let us recall that the *pair function* $\langle n, m\rangle = ((n + m)(n + m + 1)/2) + n$ is a polynomial-time computable function which is also a primitive recursive and bijective function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$.

A *decision problem* X is one whose solution is either "`yes`" or "`no`". This can be formally defined by an ordered pair (I_X, θ_X), where I_X is a language over a finite alphabet and θ_X is a total boolean function over I_X. The elements of I_X are called *instances* of the problem X.

2.1 Recognizer membrane systems

In this section, a membrane system designates any variant of P system. Recognizer membrane systems were introduced in [4] and they provide a natural framework to solve decision problems by means of devices in Membrane Computing.

Definition 1. *A membrane system Π is a recognizer membrane system if the following holds:*

1. *The working alphabet Γ of Π has two distinguished objects* `yes` *and* `no`.
2. *There exists an (input) alphabet Σ strictly contained in Γ.*
3. *The initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$ of Π are multisets over $\Gamma \setminus \Sigma$.*
4. *There exists a distinguished membrane called the input membrane.*
5. *The output region i_{out} is the environment.*
6. *All computations halt.*
7. *If $\mathcal{C}$ is a computation of Π, then either object* `yes` *or object* `no` *(but not both) must have been released into the environment, and only at the last step of the computation.*

In recognizer membrane systems any computation is either an *accepting computation* (when object `yes` is released into the environment at the last step).

For each finite multiset m over the input alphabet Σ, the *computation of the system Π with input m* starts from the configuration obtained by adding the input multiset m to the contents of the input membrane, in the initial configuration of Π. Therefore, in this kind of systems we have an initial configuration associated with each input miltiset m (over the input alphabet Σ). We denote $\Pi + m$ the membrane system Π with input multiset m.

2.2 Polynomial complexity classes of recognizer membrane systems

Next, let us recall the concept of efficient solvability by means of a family of recognizer membrane systems (see [4] for more details).

Definition 2. *A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) | n \in \mathbb{N}\}$ of recognizer membrane systems from a class $\mathcal{R}$, in a uniform way, denoted by $X \in \mathbf{PMC}_{\mathcal{R}}$, if the following statements hold:*

- *the family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$;*
- *there exists a pair (cod, s) of polynomial-time computable functions over the set I_X such that:*
  - *for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is the input multiset of the system $\Pi(s(u))$;*
  - *for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set;*
  - *the family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u)) + cod(u)$ is halting and it performs at most $p(|u|)$;*
  - *the family $\mathbf{\Pi}$ is sound with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u)) + cod(u)$, then $\theta_X(u) = 1$;*
  - *the family $\mathbf{\Pi}$ is complete with regard to (X, cod, s), that is, for each $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u)) + cod(u)$ is an accepting one.*

The polynomial complexity class $\mathbf{PMC}_{\mathcal{R}}$ is closed under polynomial-time reduction and under complement [5].

3 Tissue-like P Systems with Active Cells

This new kind of P systems keeps the inspiration keeps the foundations of classical tissue P systems, that is, the exchange of elements between the cells. Here, instead of the use of symport/antiport rules, we are going to introduce the application of the rules typically used in cell-like P systems with active membranes.

3.1 Syntax

Definition 3. *A tissue-like P system with active membranes and separation rules of degree $q \geq 1$ is a tuple $(\Gamma, \Gamma_0, \Gamma_1, H, H_0, H_1, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, where:*

- *Γ is a finite alphabet and $H = \{1, \ldots, q\}$;*
- *$\{\Gamma_0, \Gamma_1\}$ is a partition of Γ and $\{H_0, H_1\}$ is a partition of H;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over Γ;*
- *$\mathcal{R}$ is a finite set of rules over Γ of the following forms:*

- *(a)* $[a \to u]_h^\alpha$ *for* $h \in H, \alpha \in \{+,-,0\}, a \in \Gamma, u \in M_f(\Gamma)$ *(object evolution rules).*
- *(b)* $a\ [\ \ ]_h^{\alpha_1} \to [\,b\,]_h^{\alpha_2}$ *for* $h \in H, \alpha_1, \alpha_2 \in \{+,-,0\}, a, b \in \Gamma$ *(send-in communication rules).*
- *(c)* $[\ a\ ]_h^{\alpha_1} \to b\ [\ \ ]_h^{\alpha_2}$ *for* $h \in H, \alpha_1, \alpha_2 \in \{+,-,0\}, a, b \in \Gamma$ *(send-out communication rules).*
- *(d)* $[\ a\ ]_h^\alpha \to b$ *for* $h \in H, \alpha \in \{+,-,0\}, a, b \in \Gamma$ *(dissolution rules).*
- *(e)* $[\ a\ ]_h^{\alpha_1} \to [\ b\ ]_h^{\alpha_2}[\ c\ ]_h^{\alpha_3}$ *for* $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+,-,0\}, a, b, c \in \Gamma$ *(division rules for elementary membranes).*
- *(e)* $[\ a\ ]_h^{\alpha_1} \to [\ \Gamma_0\ ]_h^{\alpha_2}[\ \Gamma_1\ ]_h^{\alpha_3}$ *for* $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+,-,0\}, a \in \Gamma$ *(separation rules for elementary membranes).*
- *(f)* $[[\ \ ]_{h_0}^{\alpha_1}[\ \ ]_{h_1}^{\alpha_2}]_h^\alpha \to [[\ \ ]_{h_0}^{\alpha_3}]_h^{\alpha_5}[[\ \ ]_{h_1}^{\alpha_4}]_h^{\alpha_6}$ *for* $h, h_0, h_1 \in H$, $\alpha, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6 \in \{+,-,0\}$ *(division rules for non-elementary membranes).*
- *(f)* $[[\ \ ]_{h_0}^{\alpha_1}[\ \ ]_{h_1}^{\alpha_2}]_h^\alpha \to [\Gamma_0[\ \ ]_{h_0}^{\alpha_3}]_h^{\alpha_5}[\Gamma_1[\ \ ]_{h_1}^{\alpha_4}]_h^{\alpha_6}$ *for* $h \in H, h_0 \in H_0, h_1 \in H_1$, $\alpha, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6 \in \{+,-,0\}$ *(separation rules for non-elementary membranes).*

• $i_{out} \in H \cup \{env\}$*, where* $env \notin \Gamma \cup H$*.*

A tissue-like P system with active cells of degre $q \geq 1$ can be viewed as a set of q *cells*, labelled by elements of H, arranged in a directed structure μ given by a directed graph (the *cell* structure) whose nodes h that have $outdegree(h) = 0$ are called *elementary cells*, such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ represent the finite multisets of *objects* (symbols of the working alphabet Γ) initially placed in the q cells of the system; (b) $\mathcal{R}$ is a finite set of rules over Γ associated with the system; and (c) $i_{out} \in H \cup \{env\}$ indicates the output region. We use the term *region* i to refer to cell i in the case $i \in H$ and to refer to the "environment" of the system in the case $i = env$. If the membrane system makes no use of separation rules for non-elementary cells, then sets H_0 and H_1 will be omited. If separation rules either for elementary and non-elementary cells are not used, then we can omit either the sets H_0 and H_1 and Γ_0 and Γ_1. The *length* of a rule is the number of objects involved in it (for instance, the length of the object evolution rule $[\ a \to u\ ]_h^\alpha$ is $1 + |u|$. Let us notice that in this framework we can change (classical) object evolution rules by cooperative evolution rules (see [11] for more details).

For each cell h different for cells h with $indegree(h) \neq 0$, we denote $p(h)$ the label of the parent of h in μ. By convention, the "parent" of cells h with $indegree(h) = 0$ is the environment of the system

3.2 Semantics

An *instantaneous description* or a *configuration* $\mathcal{C}_t$ at an instant t of a P system with active cells is described by the cell structure at instant t and all multisets of objects over Γ associated with all the membranes present in the system at the moment.

An object evolution rule $[a \to u]_h^\alpha$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α in $\mathcal{C}_t$ which contains object a. When applying such a rule, object a is consumed and all objects from multiset u are produced in that membrane.

A send-in communication rule $a\,[\quad]_h^{\alpha_1} \to [\,b\,]_h^{\alpha_2}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α_1 in $\mathcal{C}_t$ such that $indegree(h) > 0$ and its parent one of its parent cells contain object a. When applying such a rule, object a is consumed from the selected parent cell and object b is produced in the corresponding cell h, and the polarization of cell h changes to α_2.

A send-out communication rule $[\ a\ ]_h^{\alpha_1} \to b\,[\quad]_h^{\alpha_2}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α_1 in $\mathcal{C}_t$ such that it contains object a. When applying such a rule, object a is consumed from such cell and object b is produced in the one of its parent cells chosen in a non-deterministic way, and the polarization of cell h changes to α_2.

A dissolution rule $[\ a\ ]_h^{\alpha} \to b$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α in $\mathcal{C}_t$, different from the output region, such that it contains object a. When applying such a rule, object a is consumed, cell h is dissolved and its objects are sent to one of the parents cells, chosen non-deterministically (or ancestors that have not been dissolved). For all h' such that $f(h') = h$ and h'' such that $f(h) = h''$, when h is dissolved, then new edges from all h'' to all h' are created, and edges from h'' to h and from h to h' are removed.

A division rule $[\ a\ ]_h^{\alpha_1} \to [\ b\ ]_h^{\alpha_2}[\ c\ ]_h^{\alpha_3}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α_1 in $\mathcal{C}_t$, different from the output region, such that it is an elementary cell and contains object a. When applying such a rule, the cell is divided into two cells with the same label, one with polarization α_1 and the other one with polarization α_2; at the same time, object a is consumed and object b appears in the first cell, and c in the second one, and the remaining objects get duplicated in the two created cells. For all h' such that $f(h') = h$ and h'' such that $f(h) = h''$, when h is dissolved, then edges from all h'' to h and from h to h' are duplicated.

A separation rule $[\ a\ ]_h^{\alpha_1} \to [\ \Gamma_0\ ]_h^{\alpha_2}[\ \Gamma_1\ ]_h^{\alpha_3}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α_1 in $\mathcal{C}_t$, different from the output region, such that it is an elementary cell and contains object a. When applying such a rule, the cell is separated into two cells with the same label, one with polarization α_1 and the other one with polarization α_2; at the same time, object a is consumed and the multiset of objects contained in membrane h get distributed: the objects from Γ_0 are placed in one cell, those from Γ_1 are placed in the second one. For all h' such that $f(h') = h$ and h'' such that $f(h) = h''$, when h is dissolved, then edges from all h'' to h and from h to h' are duplicated.

A division rule $[[\quad]_{h_0}^{\alpha_1}[\quad]_{h_1}^{\alpha_2}]_h^{\alpha} \to [[\quad]_{h_0}^{\alpha_3}]_h^{\alpha_5}[[\quad]_{h_1}^{\alpha_4}]_h^{\alpha_6}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α in $\mathcal{C}_t$, different from the output region, such that it is the parent of a cell labelled by

h_0 with polarization α_1 and of another cell labelled by h_1 with polarization α_2. When applying such a division rule to a cell labelled by h in a configuration $\mathcal{C}_t$, that cell is divided into two cells with the same label with polarizations α_5 and α_6, in such a way that the contents (multiset of objects) and relations (children and parent cells) are duplicated into the two new cells, except from cells labelled by h_0, that becomes a child cell of the first one, with polarization α_3, and h_1, that becomes a child cell of the second one, with polarization α_4. For all h' such that $f(h') = h$ and h'' such that $f(h) = h''$, when h is dissolved, then edges from all h'' to h and from h to h' are duplicated (except for edges from h to h_0 and h_1, which ones remains one for each new created cell).

A separation rule $[[\ \ ]_{h_0}^{\alpha_1}[\ \ ]_{h_1}^{\alpha_2}]_h^{\alpha} \rightarrow [\Gamma_0[\ \ ]_{h_0}^{\alpha_3}]_h^{\alpha_5}[\Gamma_1[\ \ ]_{h_1}^{\alpha_4}]_h^{\alpha_6}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a cell labelled by h with polarization α in $\mathcal{C}_t$, different from the output region, such that it is the parent of a cell labelled by h_0 with polarization α_1 and of another cell labelled by h_1 with polarization α_2. When applying such a separation rule to a cell labelled by h in a configuration $\mathcal{C}_t$, that cell is separated into two cells with the same label with polarizations α_5 and α_6, in such a way that the contents (multisets of objects) and relations (children cells) are distributed as follows: The first cell receives the multiset of objects from Γ_0, and all child cells whose label belongs to H_0; and the second cell receives the multiset of objects from Γ_1, and all child cells whose label belongs to H_1. For all h' such that $f(h') = h$ and h'' such that $f(h) = h''$, when h is dissolved, then edges from all h'' to h are duplicated, and edges from h to h' are distributed depending on whether they belong to H_0 or H_1.

In tissue-like P systems with active cells, the rules are applied according to the following principles:

- The rules associated with membranes labelled with h are used for all copies of this membrane.
- At one transition step, one object can be used by only one rule (chosen in a non-deterministic way).
- At one transition step, a *cell* can be subject of *only one* rule of types (b)–(f), and then it is applied at most once.
- Object evolution rules can be simultaneously applied to a cell with one rule of types (b)–(f). Object evolution rules are applied in a maximally parallel manner.
- If at the same time a membrane labelled with h is divided/separated by a rule of type (e) or (f) and there are objects in this cells which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, changing the objects, and then the separation is produced. Of course, this process takes only one transition step.
- Output cell can never get divided, separated, nor dissolved.

Let us consider a tissue-like P systems with active cells Π We say that configuration $\mathcal{C}_t$ yields configuration $\mathcal{C}_{t+1}$ in one *transition step*, denoted by $\mathcal{C}_t \Rightarrow_\Pi \mathcal{C}_{t+1}$, if we can pass from $\mathcal{C}_t$ to $\mathcal{C}_{t+1}$ by applying the rules from the system following the

previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that: (a) the first term is the initial configuration of the system; (b) for each $n \geq 1$, the n-th configuration of the sequence is obtained from the previous configuration in one transition step; and (c) if the sequence is finite (called *halting computation*) then the last term is a *halting configuration* (a configuration where no rule of the system is applicable to it).

All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region i_{out} associated with the halting configuration. If $\mathcal{C} = \{\mathcal{C}_t\}_{t<r+1}$ of Π ($r \in \mathbb{N}$) is a halting computation, then the *length* of $\mathcal{C}$, denoted by $|\mathcal{C}|$, is r, that is, $|\mathcal{C}|$ is the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. For each i ($1 \leq i \leq q$) we denote by $\mathcal{C}_t(i)$ the finite multiset of objects over Γ contained in all cells labelled by i (by applying division or separation rules different cells with the same label can be created) at configuration $\mathcal{C}_t$.

3.3 Families of tissue-like P systems with active cells

We use the following notations:

- $\mathcal{NAC}(\alpha, \beta, \delta)$, where $\alpha \in \{+e, -e\}$, $\beta \in \{+c, -c\}$ and $\delta \in \{+d, -d\}$, is the class of all recognizer P systems with active cells without using division nor separation rules.
- $\mathcal{DAC}(\alpha, \beta, \delta, \gamma)$, where $\alpha \in \{+e, -e\}$, $\beta \in \{+c, -c\}$, $\delta \in \{+d, -d\}$ and $\alpha \in \{+n, -n\}$,is the class of all recognizer P systems with active cells and division rules.
- $\mathcal{SAC}(\alpha, \beta, \delta, \gamma)$, where $\alpha \in \{+e, -e\}$, $\beta \in \{+c, -c\}$, $\delta \in \{+d, -d\}$ and $\alpha \in \{+n, -n\}$,is the class of all recognizer P systems with active cells and separation rules.

The meaning of parameters is the following:

- if $\alpha = +e$ (resp., $-e$) then evolution rules are permitted (resp., forbidden).
- if $\alpha = +c$ (resp., $-c$) then communication rules are permitted (resp., forbidden).
- if $\alpha = +d$ (resp., $-d$) then dissolution rules are permitted (resp., forbidden).
- if $\alpha = +n$ (resp., $-n$) then division/separation rules for elementary and non-elementary cells are permitted (resp., only division/separation rules for elementary cells are permitted).

3.4 Another (not so relevant) approach

One question discussed when this framework was being created was:

In tissue-like membrane systems, the natural definition would be the one where when we do a communication rule, the cell interacts the environment (objects go

to the environment in send-out communication rules and comes from it in send-in communication rules. The same goes to dissolution rules, that is, when a cell dissolves, its contents go to the environment.

This definition seems the best in order to capture the behavior of tissue P systems. But because of the simple structure created, it has little interest regarding the computational complexity of these systems.

If this kind of systems is defined, we can suppose that there are q cells disposed in the environment, and they can interact with it through communication and dissolution rules. But we can simulate this behavior with P systems with active membranes with $q+1$ membranes, where q membranes are situated within one that acts as the environment in the previous system. So complexity classes where these families of P systems were involved in would be weaker than classical P systems with active membranes, therefore it will not be considered.

4 Some Results About Computational Complexity

First of all, it is easy to see that every P system with active cells is at least as powerful as its active membranes counterpart. It can be proved because every P system with active membranes structure is defined by a rooted tree μ. A tree is a particular case of a graph, where cycles are not allowed. For every P system with active membranes, we can define a P system with active cells that simulates its behavior. Let $\Pi = (\Gamma, \Gamma_0, \Gamma_1, H, H_0, H_1, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ a P system with active membranes. We can create (in polynomial time) a P system with active membrane that simulates its behavior. Let $\Pi' = (\Gamma, \Gamma_0, \Gamma_1, H, H_0, H_1, \mu', \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ be the P system with active cells that simulates its behavior. μ' is constructed as follows:

- Let μ' be a single node h, where h is the label of the skin membrane of Π.
- For every membrane h' situated within another membrane h in Π, we create a node h' in μ' and add an edge from h to h'.

The directed graph obtained has the shape of a directed rooted tree, and as it has the same set of rules, semantics of the system makes Π' simulate the behavior of Π'. We can conclude with:

Theorem 1. $\mathbf{PMC}_{\mathcal{AM}(\alpha,\beta,\delta,\gamma)} \subseteq \mathbf{PMC}_{\mathcal{AC}(\alpha,\beta,\delta,\gamma)}$,

no matter which kinds of rules we are using.

4.1 Some complexity classes

As it happened with P systems with active membranes, we can use the Milano Theorem [14] to state that no computationally hard problems can be solved in polynomial time without using rules allowing the generation of an exponential number of membranes/cells in polynomial time. Then:

Theorem 2. $\mathbf{P} = \mathbf{PMC}_{\mathcal{NAC}}$

In [8, 9], an upper bound of the complexity of P systems with active membranes was given. In fact, algorithms used there did not complain about the "direction of the edges" in the graph defining the systems, so the same technique can be used here.

Theorem 3. $\mathbf{PSPACE} = \mathbf{PMC}_{\mathcal{DAC}(+e,+c,+d,+n)}$

In fact, we can use this technique to define an upper bound for systems that use separation rules instead of division rules.

Theorem 4. $\mathbf{PMC}_{\mathcal{SAM}(+e,+c,+d,+n)} \cup \mathbf{PMC}_{\mathcal{SAC}(+e,+c,+d,+n)} \subseteq \mathbf{PSPACE}$

4.2 Polarizationless P systems with active cells

In previous works, P systems with active membranes were demonstrated to be too powerful in order to obtain new frontiers to tackle the problem **P** vs. **NP**. In order to obtain less powerful systems, polarizationes were avoided, giving place to polarizationless P systems with active membranes. Some frontiers of efficiency were obtained in this new framework. We can do the same in P systems with active cells, so we would obtain *polarizationless* P systems with active cells. These systems are defined as polarizationless P systems with active membranes are defined to P systems with active membranes.

We use the following notations:

- $\mathcal{DAC}^0(\alpha, \beta, \delta, \gamma)$, where $\alpha \in \{+e, -e\}$, $\beta \in \{+c, -c\}$, $\delta \in \{+d, -d\}$ and $\alpha \in \{+n, -n\}$,is the class of all recognizer polarizationless P systems with active cells and division rules.
- $\mathcal{SAC}^0(\alpha, \beta, \delta, \gamma)$, where $\alpha \in \{+e, -e\}$, $\beta \in \{+c, -c\}$, $\delta \in \{+d, -d\}$ and $\alpha \in \{+n, -n\}$,is the class of all recognizer polarizationless P systems with active cells and separation rules.

The meaning of parameters is the same than before.

In [3, 12] that families of P systems which make no use of dissolution rules can only solve tractable problems in an efficient way. The technique used is the dependency graph technique, and we can adapt it to P systems with active cells, so:

Theorem 5. $\mathbf{P} = \mathbf{PMC}_{\mathcal{DAC}^0(+e,+c,-d,+n)} = \mathbf{PMC}_{\mathcal{SAC}^0(+e,+c,-d,+n)}$

Proof. Here, the creation of the graph differs from the original one since a cell can have two parent cells, unlike in active membranes, where each membrane could have at most one parent membrane. So, we have to contemplate this in the next algorithm:

`Input:` Π (with $\mathcal{R}$ as its set of rules and H as its label set)

$V_\Pi \leftarrow \emptyset$; $E_\Pi \leftarrow \emptyset$
`for each` rule $r \in \mathcal{R}$ of Π `do`
  `if` $r = [\ a \rightarrow u\ ]_h \wedge alph(u) = \{a_1, \ldots, a_s\}$ `then`
    $V_\Pi \leftarrow V_\Pi \cup \sum_{j=1}^{s} \{(a,h),(a_j,h)\}$
    $E_\Pi \leftarrow E_\Pi \cup \sum_{j=1}^{s} \{((a,h),(a_j,h))\}$
  `else if` $r = [\ a\ ]_h \rightarrow b\ [\ \ ]_h$ `then`
    $V_\Pi \leftarrow V_\Pi \cup \sum_{h'=f(h)} \{(a,h),(b,h')\}$
    $E_\Pi \leftarrow E_\Pi \cup \sum_{h'=f(h)} \{((a,h),(b,h'))\}$
  `else if` $r = a\ [\ \ ]_h \rightarrow [\ b\ ]_h$ `then`
    $V_\Pi \leftarrow V_\Pi \cup \sum_{h=f(h')} \{(a,h),(b,h')\}$
    $E_\Pi \leftarrow E_\Pi \cup \sum_{h=f(h')} \{((a,h),(b,h'))\}$
  `else if` $r = [\ a\ ]_h \rightarrow [\ b\ ]_h[\ c\ ]_h$ `then`
    $V_\Pi \leftarrow V_\Pi \cup \{(a,h),(b,h),(c,h)\}$
    $E_\Pi \leftarrow E_\Pi \cup \{((a,h),(b,h)),(a,h),(c,h)\}$
  `else if` $r = [\ a\ ]_h \rightarrow [\ \Gamma_0\ ]_h[\ \Gamma_1\ ]_h$ `then`
    $V_\Pi \leftarrow V_\Pi \cup \{(a,h)\}$

The running time of this algorithm is bounded by $O(|\mathcal{R}| \cdot q)$, where q is the value $max(max\{length(r) : r \in \mathcal{R}\}, |H|)$. The rest of the demonstration is similiar to the given in [3, 12].

□

In [1], a uniform solution to `QSAT` problem was given with polarizationless P systems with active membranes that make use of dissolution and division rules for elementary and non-elementary membranes. This solution, of course, can be adapted to polarizationless P systems with active cells. Thus:

Theorem 6. PSPACE = $\mathbf{PMC}_{\mathcal{DAC}^0(+e,+c,+d,+n)}$

Here, a new version of the Păun's conjecture can be outlined:

$$\mathbf{P} \stackrel{?}{=} \mathbf{PMC}_{\mathcal{DAC}^0(+e,+c,+d,-n)}$$

4.3 Minimal cooperation in polarizationless P systems with active membranes

Some interesting results have been reached in the framework of P systems with active membranes when *minimal cooperation* has been introduced. That is, with this kind of rules, we can make the objects in the regions collaborate with each other. The term *minimal* tell us that the left part of a rule can have at most two

objects, but even with this restriction, these systems are powerful enough to solve computationally hard problems.

In the context of polarizationless P systems with active cells, the following kinds of minimal cooperation in object evolution rules are considered.

- Primary minimal cooperation (**pmc**): object evolution rules are of the form $[\ u \rightarrow v\ ]_h$, where $h \in H$, u, v are multisets over Γ, and $1 \leq |u|, |v| \leq 2$, but at least one object evolution rule verifies $|u| = 2$.
- Bounded minimal cooperation (**bmc**): object evolution rules are of the form $[\ u \rightarrow v\ ]_h$, where $h \in H$, u, v are multisets over Γ, and $1 \leq |u| \leq |v| \leq 2$, but at least one object evolution rule verifies $|u| = 2$.
- Minimal cooperation and minimal production (**mcmp**): object evolution rules are of the form $[\ a \rightarrow b\ ]_h$, $[\ a \rightarrow b\ ]_h$, where $h \in H$, $a, b, c \in \Gamma$, but at least one object evolution rule is of the second type.

We use the same notations that in polarizationless P systems with active cells (that make use of classical object evolution rules), but now $\alpha \in \{pmc, bmc, mcmp\}$.

In [10], the use of *bounded minimal cooperation* were demonstrated to be strong enough to solve **NP**-complete problems.

Theorem 7. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{DAC}^0(bmc,+c,-d,-n)}$

In [13], this result was improved by using *mcmp* rules, that is:

Theorem 8. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{DAC}^0(mcmp,+c,-d,-n)}$

Nevertheless, it is different when we use separation rules instead of division rules. In this framework, it was demonstrated in [11] that the use of *bounded minimal cooperation* is not powerful enough to solve **NP**-complete problems. It was demonstrated with the algorithmic technique, and in this case we can adapt the algorithm to deal with polarizationless P systems with active cells.

Theorem 9. $\mathbf{P} = \mathbf{PMC}_{\mathcal{SAC}^0(bmc,+c,+d,+n)}$

Bearing in mind that minimal cooperation with minimal production in object evolution rules is a particular case of bounded minimal cooperation, we deduce the following result:

Theorem 10. $\mathbf{P} = \mathbf{PMC}_{\mathcal{SAC}^0(mcmp,+c,+d,+n)}$

In order to obtain efficient solutions to presumably intractable problems in the framework of polarizationless P systems with active cells, we have to make use of *primary minimal cooperation*. We can use the same solution to `SAT` problem that in [12], therefore:

Theorem 11. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{SAC}^0(pmc,+c,-d,-n)}$

5 Conclusions

In this work, we present a new kind of families of P systems, the so-called P systems with active cells. The union of the syntax and semantics of P systems with active membranes and the structure of tissue-like P systems give this kind of membrane systems some interesting properties. It is useful in order to obtain new frontiers of efficiency regarding the *direction* of the edges. We can see P systems with active membranes as directed graphs that have edges in only one direction. If we remove the restriction of the direction of the edges, we obtain P systems with active membranes. It is no surprising that these membrane systems are at least as powerful as the former ones.

Some classical results in P systems with active membranes are reviewed to obtain their equivalent when we are treating with P systems with active cells. Both systems with polarizations and polarizationless ones are studied, giving an upper bound of them and, like in the framework of active membranes, is **PSPACE**.

Minimal cooperation have been recently investigated to study its relevance in the power of polarizationless P systems with active membranes, and here we give their counterpart active cells definitions. All the results given here are quite similar to their active membranes counterparts, therefore a first question appear:

- Does the direction matter? That is, does:

$$\mathbf{PMC}_{\mathcal{AM}(\alpha,\beta,\delta,\gamma)} = \mathbf{PMC}_{\mathcal{AC}(\alpha,\beta,\delta,\gamma)}$$

  remains?

References

1. A. Alhazov, M.J. Pérez-Jiménez. Uniform solution of QSAT using polarizationless active membranes. *Machines, Computations, and Universality. Lecture Notes in Computer Science*, 4664 (2007), 122–133.
2. T.H. Cormen, C.E. Leiserson, R.L. Rivest. *An Introduction to Algorithms*. The MIT Press, Cambridge, Massachussets, 1994.
3. M.A. Gutirrez-Naranjo, M.J. Prez-Jimnez, A. Riscos-Núñez, F.J. Romero-Campero. On the power of dissolution in P systems with active membranes. *Lecture Notes in Computer Science*, 3850 (2006), 224-240.
4. M.J. Prez-Jimnez, . Romero-Jimnez, F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
5. M.J. Prez-Jimnez, . Romero-Jimnez, F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, **11**, 4 (2006), 423–434.
6. J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing* 6(4):675–695, 1977.
7. Gh. Păun. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.

8. P. Sosk. PSPACE Limits the Power of P Systems with Active Membranes. *Journal of Automata, Languages and Combinatorics*, **19** (1-4), 291–304 (2014).
9. P. Sosk, A. Rodríguez-Patón. Membrane Computing and Complexity Theory: A Characterization of PSPACE. *Journal of Computers and Systems Science*, **73**, 137–152 (2007).
10. L. Valencia-Cabrera, D. Orellana-Martn, M.. Martnez-del-Amor, A. Riscos-Nez, M.J. Pérez-Jiménez. Polarizationless P systems with active membranes: Computational complexity aspects. *Journal of Automata, Languages and Combinatorics*, 211-2 (2016), 107–123.
11. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núez, M.J. Pérez-Jiménez. Minimal cooperation in polarizationless P systems with active membranes. *Fourteenth Brainstorming Week on Membrane Computing* (BWMC2016). 327–356.
12. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núez, M.J. Pérez-Jiménez. Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. *Fundamenta Informaticae*, submitted 2016.
13. L. Valencia-Cabrera, D. Orellana-Martn, A. Riscos-Nez, M.J. Prez-Jimnez. Reaching efficiency through complicity in membrane systems: dissolution, polarization and cooperation. *Theoretical Computer Science*, submitted 2016.
14. C. Zandron, C. Ferretti, G. Mauri. Solving NP-complete problems using P systems with active membranes. *Unconventional Models of Computation*, pp. 289–301. Springer, Heidelberg (2000)

Membrane Computing Applications in Computational Economics

Eduardo Sánchez Karhunen, Luis Valencia-Cabrera

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

Summary. Major efforts have been made along the last decade on the modelling and simulation of phenomena within areas such as Biochemistry, Ecology or Robotics, providing solutions for relevant problems (signalling pathways, population dynamics or logic gene networks, or robot control and planning, among others). However, other areas initially explored have not received the same amount of attention. This is the case of computational economics, where an initial model of the so-called producer-retailer problem was proposed by Gh. and R. Păun making use of membrane computing modelling and simulation tools. In the present paper, we start designing a solution for that problem based on PDP systems, obtaining results comparable with the foundational paper. Then, an enhanced and enriched model is proposed, including several economic issues not considered in the initial model as: depreciation of production capacity, capacity increase decision mechanism, dividends payment and costs associated to production factors. Additionally, both models have been simulated making use of the framework provided by P-Lingua and MeCoSim, and delivering a custom application based on them to reproduce the virtual experiments. Finally, several scenarios have been analysed focusing on different elements included in the model.

Key words: Membrane Computing, Economy, producer-retailer problem, Computational Modelling, PDP Systems

1 Introduction

The main goal of this paper is to extend the success obtained by membrane computing as a modelling tool in different fields to a less explored one, as computational economics. In the context of the so-called producer-retailer problem, multiset rewriting rules for modelling some economic processes were proposed [10], mainly for production of goods from raw material, reception of orders from con-

sumers and purchase transactions. Also, basic numerical evolution of this system was suggested.

The paper mentioned implied a great starting point to show the capabilities of the paradigm in certain fields, but it was not focused on the reproducibility with specific conceptual and software tools. Thus, there were no indications for the reader about the specific framework within membrane computing used to obtain the results presented, neither hints about the membrane structure underlying the system nor the rest of the implementation details.

In order to reinforce the interest in Computational Economics as a promising research path within the applications of Membrane computing, the present paper details the implementation of a PDP system that replicates the results obtained by Gh. and R. Păun. We call this model "Initial producer-retailer model", explaining in depth its design in Section 3, right after introducing the context of this work in Section 2. Once obtained this first result, we propose an "Enhanced producer-retailer model" in Section 4, including several economic issues not considered in the initial model. In both cases, implementation details are provided, along with the analyses of the results obtained under different scenarios. Finally, we outline the main conclusions of this work in Section 5.

2 Preliminaries

This section starts introducing the topic of computational modelling, discussing some widely spread approaches and the choice made with membrane computing. More specifically, it will present the framework of PDP systems, used to model the economic processes presented at the end of the section.

2.1 Modelling approaches

Traditionally, biological systems have been mainly modelled using ordinary differential equations. This approach has several drawbacks: model complexity usually requires a numerical approach; model extension or improvements requires a reconstruction of the model from scratch and difficulties arise handling cases when objects appear in a reduced number of copies or processes have a strong discrete nature.

On the contrary, membrane computing [9] has many advantages for modelling systems. It presents a high degree of generality as a modelling framework (objects, multisets and evolution rewriting rules can be used to model many different situations). It is easy to add any number of membranes and/or evolution rules without essentially changing the type of P system. This modularity allows to introduce extensions or improvements to the model. Additionally, parallelism is introduced in a natural way in the model, and there are no limits to the number of variables interacting simultaneously.

Due to these previous properties, membrane computing has been applied with great success for modelling biological systems, both at a micro level, for cellular

reactions [3], and at macro level, for population dynamics [2]. Although there is a wide variety of ecosystems, they share many basic common features: there are several species interacting and a huge number of members of each one; the cyclic repetition of basic processes as feeding, growing, reproduction and death and environment influences on the system evolution.

The following section will present th framework of PDP systems, used to model the phenomena studied along this work.

2.2 Population Dynamic P systems

PDP Systems (Population Dynamic P system) were developed to consider the computational impact of the previous issues [4]. Formally, a PDP system of degree (q, m) and $T \geq 1$ units of time is a tuple $\Pi = (G, \Gamma, \Sigma, T, \{\Pi_k : 1 \leq k \leq m\}, \{E_j : 1 \leq j \leq m\}, R_E)$, where:

- $G = (V, S)$ is a directed graph with $m \geq 1$. $V = \{e_1, , e_m\}$.
- Γ and Σ are alphabets such that $\Sigma \subsetneq \Gamma$.
- $T \geq 1, n \geq 1$ are natural numbers.
- $\forall k, 1 \leq k \leq m, \Pi_k = (\Gamma, \mu, M_1, \cdots, M_q, \mathcal{R}, i_{in})$, where:
  - μ is a rooted tree with $q \geq 1$ nodes labelled with elements of $\{1, \cdots, q\} \times \{0, +, -\}$.
  - $\forall i, 1 \leq i \leq q, M_i \in M_f(\Gamma)$.
  - $\mathcal{R}$ is a finite set of rules of the type: $u[v]_i^{\alpha} \xrightarrow{p} u'[v']_i^{\alpha'}$, where $u, v, u', v' \in M_f(\Gamma), 1 \leq i \leq q, \alpha, \alpha' \in \{0, +, -\}$, and p is a probability function with domain $\{0, \cdots, T\}$. Also, the sum of probabilities of rules whose left hand side (LHS) is $u[v]_i^{\alpha}$ is 1 at each instant $t (0 \leq t \leq T)$.
  - i_{in} is a node of μ.
- $\forall j, 1 \leq j \leq m, E_j \in M_f(\Sigma)$.
- R_E is a finite set of environment rules of the type: $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j,1}}, \cdots, (y_h)_{e_{j,h}}$, where $x, y_1, \cdots, y_h \in \Sigma, \{(e_j, e_{j,i}) \in S, 1 \leq j \leq m, 1 \leq i \leq h\}$, and p_1 is a probability function with domain $\{0, \cdots, T\}$. Also, at each instant t, with $0 \leq t \leq T$, the sum of all probability function values associated to rules whose LHS is $(x)_{e_j}$ must be 1.
- There are no rules of the type $u[v]_i^{\alpha} \xrightarrow{p} u'[v']_i^{\alpha'}$ in the skin membrane of P Systems and environment rules of the type: $(x)_{e_j} \xrightarrow{p_1} (y_1)_{e_{j,1}}, \cdots, (y_h)_{e_{j,h}}$ such that $x \in u$.
- Each environment e_j contains exactly one system Π_k.

Therefore, A PDP system Π of degree (q, m) presents m environments $e_1, \cdots, e_m$ interconnected by edges of a directed graph G. Each of these environments e_j can only contain symbols of alphabet Σ, and a unique ordinary P System $\Pi_k = (\Gamma, \mu, M_1, \cdots, M_q, \mathcal{R}, i_{in})$ with this same skeleton inside each environment, but such that the initial multisets of Π_k depend on e_j and the probability functions associated with rules of Π_k depend on e_j. Finally, the semantics of PDP systems depends on the algorithm of simulation.

2.3 Economic process modeling

Traditionally, economics processes have been modelled using differential equation systems. Although these modelling techniques are predominant, many efforts have been made to investigate other techniques such as multi-agent techniques [5]. Due to the good performance of PDP systems modeling dynamics of biological systems, many researchers have proposed the idea of using membrane computing in modeling economic processes [7, 8, 1, 10, 11].

A parallelism between biological and economic processes can be identified, as Păun analyses in [10]. Many elements of membrane computing can be interpreted in economic terms. An object can represent any unit of a generic item involved in different economic processes. Also, they can represent elements of diverse nature: material good, monetary units, depreciation representation, authorization for transactions, caps of production, etc. A membrane can be any entity as producer, consumers, markets, whole economy or any other element interacting with another one. As usual in P systems, any membrane has objects associated with it. Objects can have multiplicities greater than one, so a natural way or handling them is to use multisets. Finally, multiset rewriting rules can model different kind of interactions between objects of different or the same membrane and they can represent a huge variety of processes as purchase transactions, production of goods or depreciation phenomena.

3 Initial Retailer Producer model

This section will explore in certain detail some usual economic phenomena, through the reference problem well known as "Retailer-producer" problem. After its general description, the formal model designed is presented, along with the interpretation of the elements included, the parameters involved and the deep analysis of the different modules of rules involved in the evolution of the system. Finally, the simulation results are shown and analysed.

3.1 General description

Informally, the retailer-producer problem can be described as a one good market with several players interacting with each other. A set of producers P_i that transform raw material produced by a generic source S into units of good d and a set of retailers R_j, that receive orders $\bar{d}$ from a generic consumer C. Both try to match units of d with $\bar{d}$ by means of transactions. These players are represented in 1 as circles. Each one is characterized by a parameter: P_i has a production capacity, R_j has a storage capacity, S produces raw material a at a constant rate and C generates a demand $\bar{d}$ at a constant rate. Transactions between players are represented as double arrow lines. Each of these transactions imply the exchange of monetary units characterized by its owner. u_S in possession of S, obtained from

P_i who have paid a price for each unit of a; u_i in possession of P_i, obtained from R_j who have paid a price for each unit of d; v_j in possession of C who have paid a price for each unit of $\bar{d}$ and u_C in possession of C. These are injected externally into the system allowing the consumer to throw orders.

Prices must be added to the different interactions: wholesale distributors price, that is, price at which S sells a unit of a to P_i; the price of a unit of d when sold by R_j to C and the price of a unit of d when sold by P_i to R_j. Simultaneously to the existence of prices, there are budget restrictions associated to each player. No more units of goods can be bought than the equivalent ones to the total number of monetary units owned by each player. The existence of prices enriches the evolution of the system, introducing the possibility of lack of money and making impossible to apply certain rules.

Finally, the system evolves cyclically with five steps: 1) generation of the initial conditions, 2) production of goods and reception of orders for those goods, 3) generation of purchase authorizations, 4) purchase transactions and 5) technical and cleaning rules.

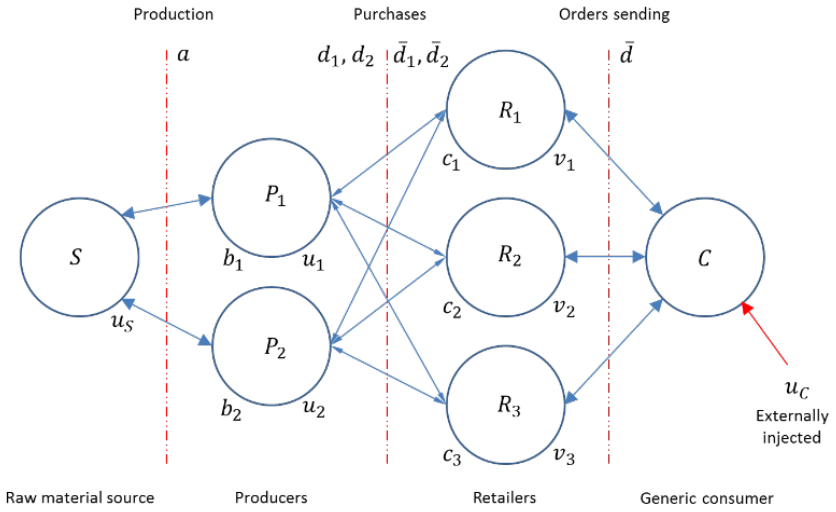


Fig. 1. Schematic representation of retailer producer problem

Production side

In economic theory, P_i has a production function (number of goods or services produced) with the following general form: $Y_i = f_i$ (factors of production). These factors are the different physical inputs used to produce goods. Typically, they are classified into three main categories: raw material, labor of workers and capital

stock. While f_i specifies how factors are transformed into goods. For simplicity, we make some assumptions. All P_i have access to the same technology, thus they all have the same production function $\forall i(Y_i = Y)$. Also, each P_i takes as factors: raw material provided by S; production capacity (also known as capital stock) and, for simplicity, labor is not considered. Thus, we can simplify the production function obtaining: $Y_i = Y = f(rawmaterial, capital) = f(a, b_i)$.

The multiplicity of a represents the total amount of raw material available for production and the multiplicity b_i represents the total production capacity of P_i. Additionally, we consider the simplest form for f, where only one unit of a and b_i are consumed to produce one unit of d. This exchange rate can be easily changed to consider more complex situations.

Demand side

In real markets, there is a bunch of individual consumers requiring units of good d. In the context of the so-called, economic rational behavior model, the behavior of each individual consumer is captured by a utility function of the form: $U_i = U_i(consumedinputs) = U_i(consumption, leisure)$. U quantifies in monetary units the happiness of individuals, making explicit their preferences about the simultaneous consumption of multiple disposable goods. Units of d obtained by consumer is called consumption and leisure can be considered as the time not dedicated to work (with a clear cost of opportunity). Classical economical models consider that rational individuals try to maximize his utility function. For simplicity, we make some assumptions. The only factor for utility is the consumption of d and labor (as complementary to leisure) is not considered. All consumers have the same utility function, same preferences and, thus, the same behavior $\forall i(U_i = U)$. This gives rise to the concept of representative consumer (more generally, representative agents). We can consider the sum of the utility functions of the population of consumers, generating a, so called, aggregate demand of d (each unit is denoted by $\bar{d}$). This can be represented as a generic consumer C. Thus, we can simplify the utility function obtaining: $U_i = U = g(\bar{d})$.

3.2 Model formalization

A simple membrane structure for the PDP system is selected with a unique environment containing one P system with two membranes. Membrane 1 is used for the R_j and P_i operations of good and order generation. Membrane 2 is used for performing the purchase transactions. The previous system will be modelled by a PDP system of degree $(2, 1)$ and $T \geq 1$ units of time $\Pi = (G, \Gamma, \Sigma, T, \mathcal{R}_E, \mu, \mathcal{R}_\Pi, \{f_r \in \mathcal{R}_\Pi\}, M_1, M_2)$, where $G = (V, E)$, with $V = \{e_1\}$ and $E = (e_1, e_1)$ and working alphabet: $\Gamma = \{b_i, d_i, u_i, c_j, \bar{d}_j, v_j, \bar{e}_j, f_(i, j) : 1 \leq i \leq k_1, 1 \leq j \leq k_2\} \cup \{R_1, R_2\} \cup \{C, S, \bar{d}, a, u_C, u_S\}$, where:

- C: aggregate generic consumer.
- S: raw material supplier.

- $\bar{d}$: unit of aggregate demand from C.
- a: unit of supplied raw material provided by S.
- u_C: monetary unit owned by C.
- u_S: monetary unit owned by S.
- b_i: unit of production capacity of $P_i, 1 \leq i \leq k_1$.
- d_i: unit of good supplied by $P_i, 1 \leq i \leq k_1$.
- u_i: monetary unit owned by $P_i, 1 \leq i \leq k_1$.
- c_j: unit of capacity of $R_j, 1 \leq j \leq k_2$.
- $\bar{d}_j$: unit of good demanded by $R_j, 1 \leq j \leq k_2$.
- v_j: monetary unit owned by $R_j, 1 \leq j \leq k_2$.
- $\bar{e}_j$: unit of good demanded by R_j and authorized for transaction unit of $\bar{d}_j, 1 \leq j \leq k_2$.
- $f_(i,j)$: authorization for $\bar{d}_j$ to be exchanged with d_i, for $1 \leq i \leq k_1, 1 \leq j \leq k_2$.
- R_1, R_2: for technical reasons.
- $\Sigma = \emptyset$.
- $R_E = \emptyset$.
- $\Pi = \{\Gamma, \mu, M_1, M_2, \mathcal{R}_\Pi\}$, where $\mu = [[\,]_2]_1$ and $M_1 = \{C, S, R_1, R_2\} \cup \{b_i^{k_{i,1}}, u_i^{k_{i,2}} : 1 \leq i \leq k_1\} \cup \{c_j^{k_{j,3}} : 1 \leq j \leq k_2\}$

Model parameters

- k_1: total number of producers.
- k_2: total number of retailers.
- k_3: units of a inserted into the system by S.
- k_4: allowed deviation from k_3.
- k_5: units of $\bar{d}$ inserted into the system by C.
- k_6: allowed deviation from k_5.
- k_7: price fixed by S for each unit of a.
- k_8: price fixed by C as an estimation of each order of good.
- $k_{i,1}$: initial production capacity of P_i, $1 \leq i \leq k_1$.
- $k_{i,2}$: initial monetary units of P_i, $1 \leq i \leq k_1$.
- $k_{j,3}$: initial capacity of R_j, $1 \leq j \leq k_2$.
- $k_{m,4}$: discrete prob distribution of units of a inserted into the system by S, $1 \leq m \leq 3$.
- $k_{m,5}$: discrete prob distribution of units of $\bar{d}$ inserted into the system by C, $1 \leq m \leq 3$.
- $k_{i,6}$: price fixed by P_i for each unit of d_i, $1 \leq i \leq k_1$.
- $k_{j,7}$: price fixed by R_j for each order of good, $1 \leq j \leq k_2$.

3.3 Modules of rules

Module 1: Initialization

The initial conditions for the cycle are generated, including the disposability of $\bar{d}$ and a. We assume that S can supply a nearly fixed amount of a at the beginning

of each cycle. To introduce some variability (not associated to any concrete real economic behavior), we will decompose the basic rule in a bunch of rules differing slightly around k_3 in the number of generated units of a. This variability is controlled by parameter k_4 and the associated probability of each rule $k_{m,4}$.

$$r_1 \equiv R_1\, s[\,]_2 \xrightarrow{p=k_{1,4}} a^{k_3+k_4} s[R_1]_2^+ \quad r_2 \equiv R_1\, s[\,]_2 \xrightarrow{p=k_{2,4}} a^{k_3} s[R_1]_2^+$$
$$r_3 \equiv R_1\, s[\,]_2 \xrightarrow{p=k_{3,4}} a^{k_3-k_4} s[R_1]_2^+ \quad r_4 \equiv R_1\, s[\,]_2 \xrightarrow{p=1-k_{1,4}-k_{2,4}-k_{3,4}} a^{k_3-2*k_4} s[R_1]_2^+$$

We also assume that C generates a nearly fixed amount of $\bar{d}$ at the beginning of each cycle. Again, we decompose the basic rule in a bunch of rules differing slightly around k_5 in the number of generated units of $\bar{d}$. This variability is controlled by parameter k_6 and the associated probability of each rule $k_(m, 5)$. C also"generates" the amount of money estimated to throw orders to R_j to be able to satisfy completely the demand $\bar{d}$, controlled by k_8.

$$r_5 \equiv R_2\, c[\,]_2 \xrightarrow{p=k_{1,5}} \bar{d}^{k_5+k_6} u_C^{(k_5+k_6)k_8} c[R_2]_2^+$$
$$r_6 \equiv R_2\, c[\,]_2 \xrightarrow{p=k_{2,5}} \bar{d}^{k_5} u_C^{k_5 k_8} c[R_2]_2^+$$
$$r_7 \equiv R_2\, c[\,]_2 \xrightarrow{p=k_{3,5}} \bar{d}^{k_5-k_6} u_C^{(k_5-k_6)k_8} c[R_2]_2^+$$
$$r_8 \equiv R_2\, c[\,]_2 \xrightarrow{p=(1-k_{1,5}-k_{2,5}-k_{3,5})} \bar{d}^{k_5-2k_6} u_C^{(k_5-2k_6)k_8} c[R_2]_2^+$$

Despite being considered in theoretical models, this idea of generating money from "nothing" at the beginning of each cycle is completely counterintuitive and do not reflects the real behavior of actual systems. This is one of the ideas that leads to an enhancement and reformulation of this initial model in following chapters.

Module 2: Producer & Retailer operation

Objects P_i have at their disposal the amount of a generated in Step 1. They compete to obtain units of a, so that they can generate units of d according to their production function. For each unit of a used by P_i it must pay a price k_7, reducing the number of u_i owned by P_i and increasing the ones u_S owned by S. Finally, each unit of d produced by P_i is denoted by d_i, with $1 \leq i \leq k$.

$$r_9 \equiv a b_i u_i^{k_7}\, c[\,]_2^+ \rightarrow u_S^{k_7} [d_i]_2^0, 1 \leq i \leq k_1$$

R_j must provide service to $\bar{d}$ generated in Step 1. They compete to get units of $\bar{d}$ to serve the demand of C. It may also be interpreted as R_j receives orders from C. For each unit ordered by C to R_j it must pay a price $k_{j,7}$, reducing the number of u_C owned by C and increasing the ones v_j owned by R_j. We will allow different prices for order to each R_j (parameter $k_{j,7}$). Each unit of good necessity $\bar{d}$ served by R_j is denoted by $\bar{d}_j$, with $1 \leq j \leq k_2$.

$$r_{10} \equiv \bar{d} c_j u_C^{k_{j,7}}\, c[\,]_2^+ \rightarrow [\bar{d}_j v_j {}_S^{k_{j,7}}]_2^0, 1 \leq i \leq k_2$$

Module 3: Performing transactions

Once orders $\bar{d}_j$ have been received by R_j and units d_i are generated by P_i, the commercial transactions can take place. One item of d is purchased by R_j from P_i to satisfy the order $\bar{d}$ carried by R_j. For each unit bought by R_j it must pay a price, reducing the number of v_j owned by R_j and increasing the ones u_i owned by P_i. Capacities c_j and b_i consumed in the production of d_i and $\bar{d}_j$ are set free.

$$[d_i \bar{d}_j v_j^{price}]_2 \xrightarrow{probability} [b_i c_j u_i^{price}]_2$$

Additionally, we can associate a probability to each possible transaction comprising many effects: the confidence of R_j on P_i; the price of the product offered by P_i; the willing of R_j to buy a good or the necessity of P_i to sell a good. Depending on the effects considered and their variability during the process, these probabilities must be computed once at the beginning of the process or recalculated after each cycle. An intuitive way of thinking about these probabilities is that the unit probability is distributed among a bunch of rules of this type:

$$[d_1 \bar{d}_1 v_1^{price}]_2 \xrightarrow{p_{11}} [b_1 c_1 u_1^{price}]_2 \qquad [d_2 \bar{d}_1 v_1^{price}]_2 \xrightarrow{p_{12}} [b_2 c_1 u_2^{price}]_2$$
$$[d_3 \bar{d}_1 v_1^{price}]_2 \xrightarrow{p_{13}} [b_3 c_1 u_3^{price}]_2$$

so that $p_{11} + p_{12} + p_{13} = 1$. On the other hand, PDP systems do not allow a direct translation of rules of this type because the evolution of any possible LHS is determined by a set of rules summing probability one. This drawback is solved creating for each exchange transaction between P_i and R_j, a symbol $f_{j,i}$ that acts as an authorization card for the transaction. This $f_{j,i}$ follows the originally desired probability distribution and can be used to simulate geographical barriers between players or preference for one of the producers. Now, probabilities associated to rules with the same LHS sum up to one and the purchase transactions can take place but now with probability one.

$$r_{14} \equiv [\bar{d}_1]_2 \xrightarrow{p=1} [\bar{d}_1 f_{1,1}]_2 \qquad r_{15} \equiv [\bar{d}_1]_2 \xrightarrow{p=0} [\bar{d}_1 f_{1,2}]_2$$
$$r_{16} \equiv [\bar{d}_2]_2 \xrightarrow{p=0.5} [\bar{d}_2 f_{2,1}]_2 \qquad r_{17} \equiv [\bar{d}_2]_2 \xrightarrow{p=0.5} [\bar{d}_2 f_{2,2}]_2$$
$$r_{18} \equiv [\bar{d}_3]_2 \xrightarrow{p=0.15} [\bar{d}_3 f_{3,1}]_2 \qquad r_{20} \equiv [\bar{d}_3]_2 \xrightarrow{p=0.85} [\bar{d}_3 f_{3,2}]_2$$

$$r_{20} \equiv [d_i \bar{d}_j f_{j,i} v_j^{k_{2,6}}]_2^0 \to [b_i c_j u_i^{k_{2,6}}]_2^-, 1 \le i \le k_1, 1 \le j \le k_2$$

Further developments of the model could explore the possibility of considering dynamic probabilities for these transactions.

Technical & cleaning rules

Finally, some rules are necessary for technical reasons. $f_{i,j}$ not exhausted in purchase transactions have no utility and symbols r_1 and r_2 are restored to their original location.

$$r_{26} \equiv [f_{i,j}]_2^- \rightarrow []_2^0, 1 \leq i \leq k_1, 1 \leq j \leq k_2$$
$$r_{30} \equiv [r_1 r_2]_2^- \rightarrow r_1 r_2 []_2^0$$

d_i, $\bar{d}_j$ and v_j not exchanged represent real things and cannot be cleaned.

$$r_{27} \equiv [\bar{d}_j]_2^- \rightarrow \bar{d}_j\,[]_2^0, 1 \leq j \leq k_2 \qquad r_{28} \equiv [v_j]_2^- \rightarrow v_j []_2^0, 1 \leq j \leq k_2$$
$$r_{29} \equiv [d_i]_2^- \rightarrow d_i\,[]_2^0, 1 \leq i \leq k_1$$

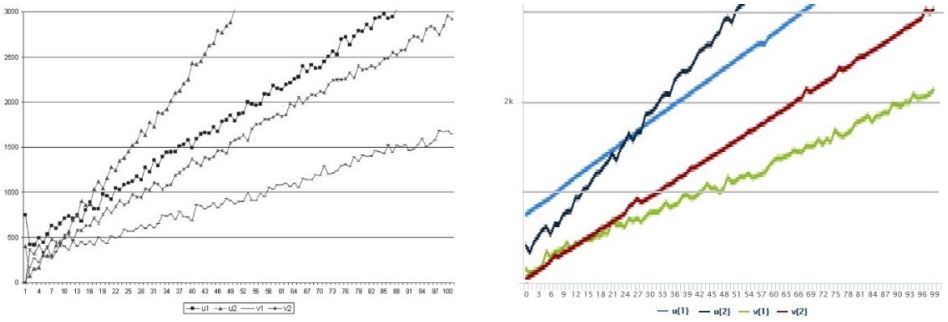
Similarly, we have the symmetric operations at the beginning of the cycle, pushing these elements into the operational membrane 2.

$$r_{12} \equiv [\bar{d}_j]_2^+ \rightarrow \bar{d}_j\,[]_2^0, 1 \leq j \leq k_2 \qquad r_{13} \equiv [v_j]_2^+ \rightarrow v_j []_2^0, 1 \leq j \leq k_2$$
$$r_{11} \equiv [d_i]_2^+ \rightarrow d_i\,[]_2^0, 1 \leq i \leq k_1$$

3.4 Simulation results

To compare the results of our implementation to the ones described by Paun, we will try to use the same set of values and number of cycles. In our model, 200 cycles with 5 steps in each cycle using DNDP-4 algorithm as inference engine.

For this purpose we consider the chart provided in Pauns article and compare it to the plot of the same output variables in our model (u_1, u_2, v_1, v_2). Both charts are represented with the same axis scale and number of T cycles.



(a) Pauns Model (b) Initial Model

Fig. 2. Evolution of monetary units owned by retailers - Paun Model vs Initial Model

4 Enhanced model

After the initial model presented, it started a process to go deeper into economic phenomena, leading to an enhanced model, enriched with a number of features of

Parameter	Value/s	Description
k_1	2	Total number of producers
k_2	3	Total number of retailers
k_3	60	Units of a inserted into the system by S
k_4	1	Deviation from k_3
k_5	60	Units of $\bar{d}$ inserted into the system by C
k_6	1	Deviation from k_5
k_7	11	Price fixed by S for each unit of a
k_8	14	Price fixed by C as an estimation of each order of good
$k_{i,1}$	$(65, 35)$	Initial production capacity of P_i, $1 \leq i \leq k_1$
$k_{i,2}$	$(750, 400)$	Initial monetary units of P_i, $1 \leq i \leq k_1$
$k_{j,3}$	$(50, 30, 20)$	Initial capacity of R_j, $1 \leq j \leq k_2$
$k_{m,4}$	$(0.01, 0.95, 0.03)$	Prob distrib. of a inserted into the system by S
$k_{m,5}$	$(0.03, 0.90, 0.04)$	Prob distrib. of units of $\bar{d}$ inserted into the system by C
$k_{i,6}$	$(12, 13)$	Price fixed by P_i for each unit of d_i
$k_{j,7}$	$(13, 14, 15)$	Price fixed by R_j for each order of good j, $1 \leq j \leq k_2$

Table 1. MeCoSim simulation parameter values

interest. This section focuses in the description of the new ingredients involved, the new processes studied and the details of the new model designed, following the section a structure similar to the previous one, from the foundations about the processing to the exploration of the model and analysis of the simulations.

4.1 General description

The behavior of the previous initial model can be condensed as follows: a steady increase of monetary units owned by P_i, R_j and C; nearly stable P_is and R_j's capacities and monetary units obtained by S get out of circulation in the system. These facts can be explained by the absence of variations in the rest of parameter of the model. First, prices associated to goods, raw material and aggregate demand are initially settled and remain unchanged during the system evolution (absence of a natural process trying to find a price of equilibrium). Secondly, P_i's and R_j's capacities are fixed and no changes are allowed (lack of a capital market). Additionally, there is an artificial exogenous injection of monetary units at the beginning of each cycle. To get our initial model nearer to real situations, new issues must be modelled. In the next chapters, some of the latter restrictions

will be relaxed or modified. First, variations of P_i's and R_j's capacities will be allowed (associated to capital stock depreciation and investment decisions). Secondly, the external injection of monetary units will be substituted by new cyclic monetary flows in the systems. Finally, randomness arise in a much more elegant way, characteristic of PDP models. This enhanced model is represented in Fig. 3.

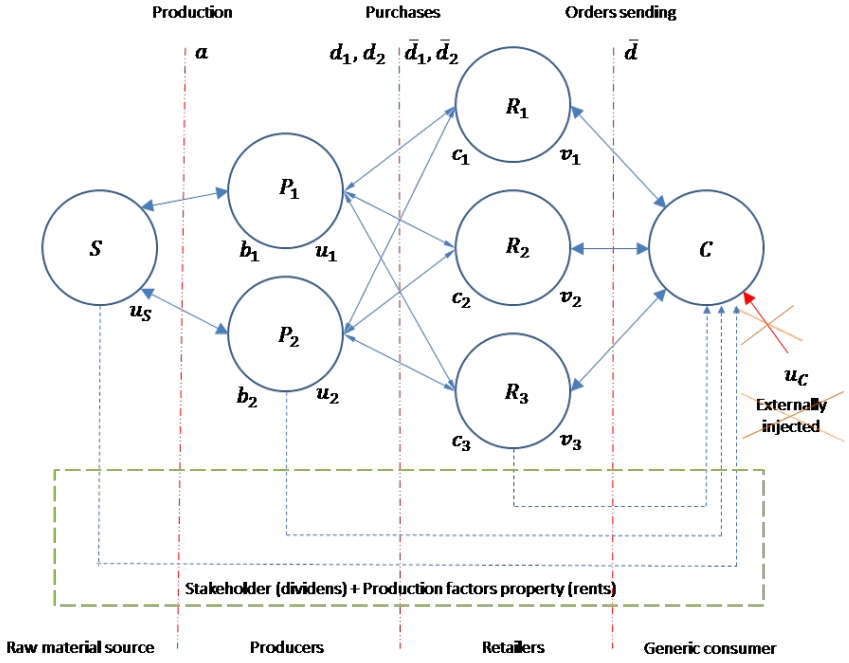


Fig. 3. Schematic representation of retailer producer problem - Enhanced Model

Randomness in a PDP-way

In our first model, the randomness of a and $\bar{d}$ was introduced in a very naive way. Making use of a bunch of rules to generate that random behavior where each rule represented a small variations around a central value. However, a more elegant "PDP-way" of generating randomness in rewriting rules has been proposed. Consider a set of generic rules with the following structure:

$$[s] \rightarrow [sa^{N-L}w^{2L}] \qquad [w] \xrightarrow{0.5} [\#] \qquad [w] \xrightarrow{0.5} [a]$$

Its willing is to generate approximately N units of a (a^N). First, we generate $a^{N-L}w^{2L}$, where, $N-L$ represents the lower limit of a hypothetic range for multiplicity of a and $N+L$, its upper limit. Secondly, two possible rules are applied to this new symbol w, each one with probability 0.5. Transforming w in one unit of

a or clearing it. The range of possible values for multiplicity of a is $[a^{N-L}, a^{N+L}]$. In our enhanced model, this strategy will be used at the beginning of each cycle in the amount of a generated by S, of $\bar{d}$ produced by C and in the investment decision mechanism. Further developments of the model could consider more sources of variability: in prices, in the probabilities of performing transactions between P_i and R_j.

Ownership of production factors and stakeholders

In real situations, there are no external injections of monetary units into the system to maintain it evolving. On the contrary, real economy dynamically adjust its parameters internally to maintain its activity cycle after cycle. Hence, we must consider in our model some alternatives to replace this artificially injected money. In a typical macroeconomic model, factors are property of the aggregate consumer C. Thus, P_i (and R_j as intermediate producers) must hire these factors out from its owners paying an amount of money for them (producers costs). In our model, there are only costs associated to production capacities. Secondly, in a typical economy, C is a stakeholder of P_i and R_j. Thus, the initial number of monetary units in possession of P_i and R_j can be interpreted as the initial investment of C. At the end of each cycle, stakeholders expect receiving a certain amount of dividend depending on the benefits obtained by the company. Benefits not distributed remain in the company allowing to pay costs of factors. These two mechanisms generate a flow of monetary units from u_i and v_j to u_C. Finally, to make our system closed with no external factors or agents acting on it, C must also be stakeholder of S. For simplicity, in our model, S does not need any production capacity to generate units of a (no production costs). Therefore, there will be a flow of monetary units from u_S to u_C. Provided these three sources of monetary units for u_C, there is no more need of an external injection of monetary units. In this enhanced model, the total number of monetary units flowing in the system is constant, and transactions between P_i, R_j, S and C creates monetary unit flows preventing them from accumulation.

Investment decision capacity increase

Once purchase transactions have been performed, P_i have probably obtained a surplus. Although in our initial model they simply accumulated these monetary units, in real situations they must decide what to do with their earnings. This is known as the investment saving decision. There are two choices: to dedicate part of it to accumulate more production capacity. In other words, take decisions about capital stock increase. Or instead of it, remain capacity unchanged, leaving earnings accumulated as savings. This decision, should be based on concrete facts.

For this purpose, we will extend the utility of the "authorization" system ($f_{j,i}$) created in the initial model. Basically, not exhausted authorizations will be interpreted as the existence of demand from R_j not satisfied. In an ideal situation,

these $f_{j,i}$ are consumed while purchase transactions are performed. However, in some cases transactions cannot be performed but symbol $f_{j,i}$ is present (aborted transaction). If it is due to the lack of capacity, it should be increased. Otherwise, it remains unchanged.

Capital stock depreciation

Production capacity suffers a phenomenon called depreciation. There exist several economic interpretations for this depreciation: obsolescence; a reduction in the remaining value of future goods this capital stock can produce or a reduction of the market price of capital. In macroeconomic theory, the behavior of capital stock is: $K_t = K_{t-1} - D_{t-1} + I_t$, where: K_t is the capital stock value (production capacity) at time t; K_{t-1} is the capacity at time $t-1$; D_{t-1} is the depreciation of K_{t-1} and I_t is the inversion at time t. Thus, a mechanism for increasing capacity is needed to ensure recovery from depreciation if needed to satisfy orders not exhausted. For simplicity, we assume a constant depreciation rate: $D_{t-1} = \delta K_{t-1}$. Thus, the previous equation can be written as: $K_t = (1-\delta)K_{t-1} + I_t$. This depreciation can be modelled as a fixed reduction of the multiplicity of b_i.

4.2 Model formalization

The initial model must be modified to consider the new phenomena considered. New symbols are added to the working alphabet, mainly associated to the new way of random generation (p, q, m_i) and the capacity increase mechanism (g_i, y_i, z_i); meanwhile, other ones are eliminated due to the creation of monetary flows in the system (u_S, R_2): $\Gamma_{extended} = \Gamma / \{u_S, R_2\} \cup \{g_i, y_i, , m_i, z_i, h_i : 1 \leq i \leq k_1\} \cup \{p, q\}$.

Additionally, the set of rules suffer changes: a) the generation of a and $\bar{d}$ is adapted to PDP mechanism of randomness, b) P_i and R_j operation are slightly modified to consider Cs property of raw material source, c) new rules to consider payments for capacity and clearing of non-paid capacity units and, also, for capacity increase mechanism and dividend distribution and d) purchase transaction rules are adapted for the following steps of capacity depreciation. Also, new necessary parameters are considered in the model meanwhile other ones are given a new interpretation and other are no longer needed.

It is no necessary to modify the original membrane structure in our new model. Finally, this modified system will be modelled by a PDP system of degree $(2, 1)$ and $T \geq 1$ units of time $\Pi = (G, \Gamma, \Sigma, T, R_E, \mu, \mathcal{R}_\Pi, \{f_r \in \mathcal{R}_\Pi\}, M_1, M_2)$, where $G = (V, E)$, with $V = \{e_1\}$ and $E = \{(e_1, e_1)\}$, and working alphabet: $\Gamma = \{b_i, d_i, u_i, c_j, \bar{d}_j, v_j, \bar{e}_j, f_{j,i}, g_i, y_i, z_i, m_i, h_i : 1 \leq i \leq k_1, 1 \leq j \leq k_2\} \cup \{R_1\} \cup \{C, S, \bar{d}, a, u_C, p, q\}$, where:

- C: aggregate generic consumer.
- S: raw material supplier.
- a: unit of supplied raw material provided by S.

- p: randomness generator for a provision by S.
- $\bar{d}$: unit of aggregate demand from C.
- q: randomness generator for $\bar{d}$ generation by C.
- u_C: monetary unit owned by C.
- b_i: unit of production capacity of P_i, $1 \leq i \leq k_1$.
- h_i: unit of production capacity of P_i before depreciation, $1 \leq i \leq k_1$.
- d_i: unit of good supplied by P_i, $1 \leq i \leq k_1$.
- u_i: monetary unit owned by P_i, $1 \leq i \leq k_1$.
- c_j: unit of capacity of R_j, $1 \leq j \leq k_2$.
- $\bar{d}_j$: unit of good demanded by R_j, $1 \leq j \leq k_2$.
- v_j: monetary unit owned by R_j, $1 \leq j \leq k_2$.
- $\bar{e}_j$: unit of good demanded by R_j and authorized for transaction, $1 \leq j \leq k_2$.
- $f_(i,j)$: authorization for $\bar{d}_j$ to be exchanged with d_i, $1 \leq i \leq k_1$, $1 \leq j \leq k_2$.
- y_i: unit (in idle state) of aborted purchase transactions considered for capacity increase, $1 \leq i \leq k_1$.
- m_i: randomness generator for y_i, $1 \leq i \leq k_1$.
- z_i: activated unit of aborted purchase transactions considered for capacity increase, $1 \leq i \leq k_1$.
- R_1: for technical reasons.
- g_i: for technical reasons, $1 \leq i \leq k_1$.
- $\Sigma = \emptyset$.
- $R_E = \emptyset$.
- $\Pi = (\Gamma, \mu, M_1, M_2, \mathcal{R}_\Pi, \{f_r \in \mathcal{R}_\Pi\})$, where $\mu = [[\,]_2]_1$, $M_1 = \{C, S, R_1\} \cup \{g_i, u_i^{7k_{i,1}k_{10}} : 1 \leq i \leq k_1\} \cup \{v_j^{7k_{j,3}k_{10}} : 1 \leq j \leq k_2\}$ and $M_2 = \{c_j^{k_{j,3}} : 1 \leq j \leq k_2\} \cup \{b_i^{k_{i,1}} : 1 \leq i \leq k_1\}$

Model parameters

In comparison with the previous initial model, we have introduced modifications in the set of parameters. Some of them have been modified in terms of their meaning, while others have been simply relabeled. Finally, the set of parameters for our model is:

- k_1: total number of producers.
- k_2: total number of retailers.
- k_3: raw material inserted into the system by S min value of range.
- k_4: raw material inserted into the system by S max value of range.
- k_5: aggregate demand inserted into the system by C min value of range.
- k_6: aggregate demand inserted into the system by C max value of range.
- k_7: price fixed by S for each unit of a.
- k_8: number of failed purchases considered for increasing capital stock min value.
- k_9: number of failed purchases considered for increasing capital stock max value.

- k_{10}: cost of capital stock per cycle.
- k_{11}: depreciation rate of capital stock.
- k_{12}: step of capacity increase.
- k_{13}: dividend percentage.
- $k_{i,1}$: initial production capacity of P_i, $1 \leq i \leq k_1$.
- $k_{i,2}$: price fixed by P_i for each unit of d_i, $1 \leq i \leq k_1$.
- $k_{j,3}$: initial capacity of R_j, $1 \leq j \leq k_2$.
- $k_{j,6}$: price fixed by R_j for each order of good j, $1 \leq j \leq k_2$.

4.3 Modules of rules

Based on the cyclic evolution of the initial model we have expanded it including the new operations (Fig. 4): 1) generation of aggregate demand and raw material disposability and rents payment for P_i's and R_j's capacity, 2) production of goods and reception of orders, 3) generation of the authorizations for purchase transactions, 4) purchase transactions and 5) capacity increase decision, capacity depreciation and dividend payment.

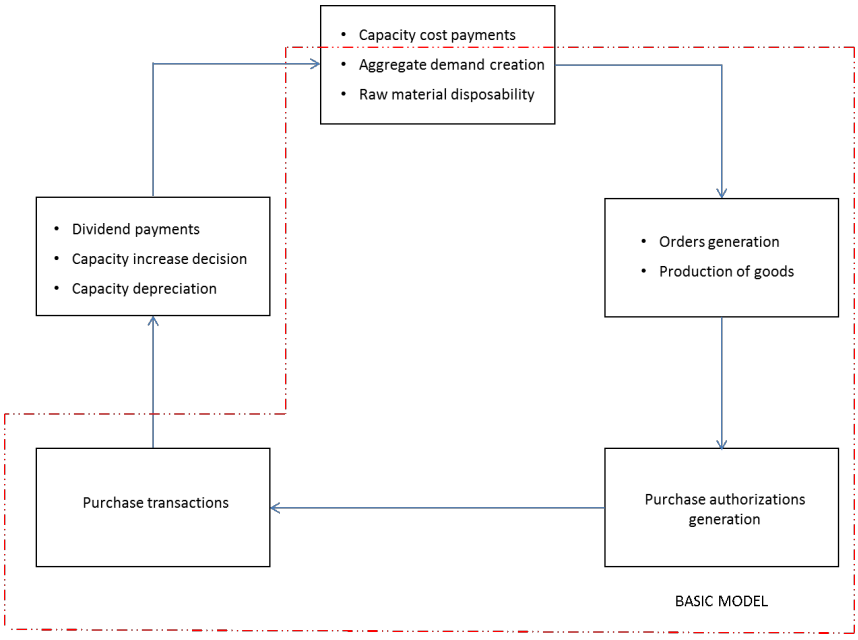


Fig. 4. Modules of rules Enhanced Model

Module 1: Production factor and demand generation

Our first step consists in generating the initial conditions of the cycle: restoring aggregate demand and ensuring raw material disposability. Unlike the initial model, we will use the PDP-way of generating randomness in the amount of generated a and $\bar{d}$. Using new symbols p (respectively, q) to control the range $[k_3, k_4]$ (respectively, $[k_5, k_6]$) of possible values of a (respectively, $\bar{d}$). This operation can be performed with a simple set of rules:

$$r_1 \equiv R_1 sc\,[\,]_2^- \rightarrow a^{k_3} p^{k_4-k_3} \bar{d}^{k_5} q^{k_6-k_5} sc\,[\,]_2^+$$

$$r_2 \equiv p\,[\,]_2^- \xrightarrow{p=0.5} a\,[\,]_2^+ \qquad r_4 \equiv q\,[\,]_2^- \xrightarrow{p=0.5} [\,]_2^+$$
$$r_3 \equiv p\,[\,]_2^- \xrightarrow{p=0.5} [\,]_2^+ \qquad r_5 \equiv q\,[\,]_2^- \xrightarrow{p=0.5} \bar{d}\,[\,]_2^+$$

Additionally, The generation of a and $\bar{d}$ is unified in a single rule but rules on p and q remain separated allowing their independent random behaviour. Also, any reference to the spontaneous appearance of monetary units is removed from the rules taking place at this step.

Module 2: Producers costs

The idea of Cs property of factors means that R_j and P_i must pay, at the beginning of each cycle, for their capacities c_j and b_i. For simplicity, a unique common parameter k_{10} has been selected for both capacities cost. For each unit of b_i used by P_i it must pay a price, reducing the number of u_i owned by P_i and increasing the u_C owned by C. Similarly, for each unit of c_i used by R_j it must pay a price, reducing the number of v_j owned by R_j and increasing the ones u_C owned by C. If R_j and P_i do not have enough monetary units to pay for their capacities, they must give up using them and restore the value of each of these capacity units to their proprietaries via u_C units.

$$r_9 \equiv u_i^{k_{10}}\,[b_i]_2 \rightarrow b_i u_C^{k_{10}}\,[\,]_2^+, 1 \leq i \leq k_1 \qquad r_9 \equiv v_j^{k_{10}}\,[c_j]_2 \rightarrow c_j u_C^{k_{10}}\,[\,]_2^+, 1 \leq j \leq k_2$$

$$r_{11} \equiv [b_i]_2^+ \rightarrow u_C^{k_{10}}[\,]_2, 1 \leq i \leq k_1 \qquad r_{11} \equiv [c_j]_2^+ \rightarrow u_C^{k_{10}}[\,]_2, 1 \leq j \leq k_2$$

Module 3: Producers & retailers operations

P_is rules are slightly modified to include another big conceptual change of our new model: C is the owner of S. This idea means a revolution in the system: in the initial model, S simply accumulated the monetary units u_S. Now, these units travel from P_i to S and, again, return to C. For each unit of a used by P_i it must pay a price (k_7 monetary units), reducing the number of u_i owned by P_i and increasing the ones u_C owned by C.

$$r_1 4 \equiv ab_i u_i^{k_7} [\,]_2^+ \rightarrow u_C^{k_7} [d_i]_2^0, 1 \leq i \leq k_1$$

R_j must face to an amount of aggregate demand generated previously competing to catch units of $\bar{d}$ to serve necessities of C. It also can be interpreted as R_j receives orders from C. We will allow each retailer to fix a different price. For each unit of $\bar{d}$ ordered by C to R_j it must pay a price $k_{j,6}$, reducing the number of u_C owned by C and increasing the ones v_j owned by R_j. Finally, capacity units not consumed are transferred out of membrane 1, waiting for later depreciation operations.

$$r_{15} \equiv \bar{d} c_j u_C^{k_{j,6}} [\,]_2^+ \rightarrow [\bar{d}_j v_j^{k_{j,6}}]_2^0, 1 \leq j \leq k_2$$
$$r_{16} \equiv b_i [\,]_2 \rightarrow [b_i]_2, 1 \leq i \leq k_1$$
$$r_{17} \equiv c_j [\,]_2 \rightarrow [c_j]_2, 1 \leq j \leq k_2$$

Module 4: Purchase transactions

This module remains almost unchanged with respect to the initial model. A first step of generation of transaction authorizations. Once generated, we can perform the purchase transactions but now with probability one. Again, the discrete probability distributions are embedded in the authorizations generation rules. One item of d_i is purchased by R_j from P_i to satisfy the order $\bar{e}_j$ carried by R_j. For each unit of d_i, R_j must pay a price, reducing the number of v_j and increasing the ones u_i owned by P_i. Capacities c_j and b_i consumed producing d_i and $\bar{e}$ are freed. Finally, free b_i are transformed into new symbols h_i waiting for depreciation operations.

$$r_{18} \equiv [\bar{d}_1]_2 \xrightarrow{p_{1,1}=1} [\bar{e}_1 f_{1,1}]_2 \qquad r_{19} \equiv [\bar{d}_1]_2 \xrightarrow{p_{1,2}=1} [\bar{e}_1 f_{1,2}]_2$$
$$r_{20} \equiv [\bar{d}_2]_2 \xrightarrow{p_{2,1}=0.5} [\bar{e}_2 f_{2,1}]_2 \qquad r_{21} \equiv [\bar{d}_2]_2 \xrightarrow{p_{2,2}=0.5} [\bar{e}_2 f_{2,2}]_2$$
$$r_{22} \equiv [\bar{d}_3]_2 \xrightarrow{p_{3,1}=0.15} [\bar{e}_3 f_{3,1}]_2 \qquad r_{23} \equiv [\bar{d}_3]_2 \xrightarrow{p_{3,2}=0.85} [\bar{e}_3 f_{3,2}]_2$$

$$r_{24} \equiv [d_i \bar{e}_j f_{j,i} v_j^{k_{i,2}}]_2 \rightarrow u_i^{k_{i,2}} [h_i c_j]_2^-, 1 \leq i \leq k_1, 1 \leq j \leq k_2$$

Further developments of the model could explore how to consider dynamic probabilities for these transactions.

Module 5: Dividends distribution

Once purchase transactions have been performed, the remaining monetary units owned by R_j and P_i can be interpreted as their benefits. On the other hand, C can be interpreted as stakeholder of R_j and P_i, and their initial monetary units can be considered the amount of money already invested by them. In this context, a dividend payment can be considered. This dividend percentage is controlled by parameter k_{13}. For simplicity, this will be considered only in P_i:

$$r_{25} \equiv [v_j]_2^- \rightarrow v_j[\,]_2^0, 1 \leq j \leq k_2$$
$$r_{26} \equiv [u_i]_2^- \xrightarrow{p=k_{13}} u_C\,[\,]_2^0, 1 \leq i \leq k_1$$
$$r_{27} \equiv [u_i]_2^- \xrightarrow{p=1-k_{13}} u_i\,[\,]_2^0, 1 \leq i \leq k_1$$

Module 6: Capacity depreciation

As explained in previous chapter, depreciation will be considered as a reduction of production capacity. For simplicity, it will only be considered a P_i's capacity depreciation. This can be easily modelled as a reduction of b_i's multiplicity with a probability controlled by parameter k_{11}, representing capacity disappearance rate.

$$r_{31} \equiv [h_i]_2^- \xrightarrow{p=1-k_{11}} [b_i]_2^0, 1 \leq i \leq k_1 \qquad r_{32} \equiv [h_i]_2^- \xrightarrow{p=k_{11}} [\,]_2^0, 1 \leq i \leq k_1$$

The global evolution suffered by b_i can be outlined in the following flow:

$$[b_i]_2^0 \xrightarrow{payrents} b_i[\,]_2^+ \xrightarrow{productionofgoods} [h_i]_2^- \xrightarrow{depreciationrules} [b_i]_2^0$$

Further developments of the model could extend the depreciation rules to all actors' capacities of the system. Indeed, the application of this depreciation rules to any capacity is paired to the necessity of a production capacity increase decision mechanism. If only depreciation acts, it will be reached a point of capacity exhaustion that stops system evolution.

Module 7: Capacity increase decision

The number of aborted transactions considered to increase capacity is controlled by the multiplicity of a new symbol m_i. This will be arbitrarily low to generate a reasonably rate of capacity increasing. Additionally, randomness will be included in the generation of symbol m_i.

In the previous sections, we analysed the circumstances accompanying a non-performed authorized purchase transaction and determined that this could be a good signal to trigger a capacity increase mechanism. If it is not due to a lack of producer capacity, it is not necessary to increase it.

$$r_{28} \equiv [f_{j,i}d_i]_2^- \rightarrow \quad [d_i]_2^0,\ 1 \leq i \leq k_1, 1 \leq j \leq k_2$$
$$r_{29} \equiv [f_{j,i}h_i]_2^- \xrightarrow{1-k_{11}} [b_i]_2^0,\ 1 \leq i \leq k_1, 1 \leq j \leq k_2$$
$$r_{30} \equiv [f_{j,i}h_i]_2^- \xrightarrow{k_{11}} \quad [\,]_2^0, \quad 1 \leq i \leq k_1, 1 \leq j \leq k_2$$

Otherwise, it is necessary to increase it (controlled by parameter k_{12}). To ensure a gradual adaptation of capacity it will be introduced a limit to the number of considered aborted transactions (represented by multiplicity of symbol y_i). Additionally, to incorporate randomness into the process, rules in the PDP-way will be included for the generation of symbol y_i, using symbol m_i. The range of values for

y_i varies in range $[k_8, k_9]$. Symbol z_i is simply an evolved form of y_i to determine the exact moment of activating this operation. Finally, non-exhausted units of $f_{j,i}$ and z_i are removed.

$$r_6 \equiv g_i[\,]_2^0 \rightarrow [g_i y_i^{k_8} m_i^{k_9-k_8}]_2^+, 1 \leq i \leq k_1$$
$$r_7 \equiv [m_i]_2^+ \xrightarrow{0.5} [\,]_2^0, 1 \leq i \leq k_1 \qquad r_8 \equiv [m_i]_2^+ \xrightarrow{0.5} [y_i]_2^0, 1 \leq i \leq k_1$$
$$r_{33} \equiv [y_i]_2^- \rightarrow [z_i]_2^0, 1 \leq i \leq k_1$$
$$r_{34} \equiv [f_{j,i} z_i]_2^0 \rightarrow b_i^{k_{12}}[\,]_2^+, 1 \leq i \leq k_1, 1 \leq j \leq k_2$$
$$r_{35} \equiv [f_{j,i}]_2^+ \rightarrow [\,]_2^0, 1 \leq i \leq k_1, 1 \leq j \leq k_2 \qquad r_{36} \equiv [z_i]_2^+ \rightarrow [\,]_2^0, 1 \leq i \leq k_1$$

Technical & cleaning rules

Finally, some rules are necessary for technical reasons. $\bar{e}_j$ and v_j not exchanged represent real received orders and monetary units so cannot be eliminated.

$$r_{13} \equiv v_j[\,]_2^+ \rightarrow [v_j]_2^0, 1 \leq j \leq k_2 \qquad r_{37} \equiv [\bar{e}_j]_2^+ \rightarrow [\bar{d}_j]_2^0, 1 \leq j \leq k_2$$

Symbols r_1 and g_i are restored to their initial location. r_1 controls the generation of a and $\bar{d}$, while g_i controls the generation of symbols y_i.

$$r_{38} \equiv [r_1]_2^- \rightarrow r_1[\,]_2^0 \qquad r_{39} \equiv [g_1]_2^- \rightarrow g_1[\,]_2^0, 1 \leq j \leq k_2$$

4.4 Simulation results

To make these results comparable with the ones obtained from the previous model, we will use a similar set of values. The complete relation of parameters is:

Simulations will be performed (again 200 cycles with 5 steps in each cycle using DNDP-4 algorithm as inference engine) for different situations to show the effect of each phenomenon included in the model, along with its contribution to the global behavior and stability of the system. In the following section, several situations are discussed. Case A: (capacity depreciation standalone) we will show producers capacity evolution when depreciation rate = 0.1 and capacity increase mechanism is deactivated. Initial capacity of producers has been set to $k_{1,1} = 65$ and $k_{2,1} = 35$. As expected, along the cycles capacities are reduced until they are completely exhausted. The slope of these curves is controlled by k_{11}. As seen in the previous chapters, a mechanism of capacity increase is necessary to maintain the evolution of the system.

Case B (capacity depreciation + capacity increase mechanism): we will show how the previous evolution changes when capacity increase mechanism and dividend payment mechanisms are activated. These mechanisms parameters are $k_{12} = 1$ (step of capacity increase), $k_8 = 3$ and $k_9 = 5$ (range of aborted purchase transaction considered), and $k_{13} = 0.01$ (dividend percentage). As expected, in a cycle each producers producer capacities suffer depreciation. This diminishing of production generates the abortion of multiple purchase transactions that activate

Parameter	Value/s	Description
k_1	2	Total number of producers
k_2	3	Total number of retailers
k_3	59	Units of a inserted into the system by S min value of range
k_4	62	Units of a inserted into the system by S max value of range
k_5	59	Units of $\bar{d}$ inserted into the system by C min value of range
k_6	62	Units of $\bar{d}$ inserted into the system by C max value of range
k_7	11	Price fixed by S for each unit of a
k_8	3	# failed purchases considered for increasing capital min value
k_9	5	# failed purchases considered for increasing capital max value
k_{10}	2	cost of capital stock per cycle
k_{11}	0.1	depreciation rate of capital stock
k_{12}	1	step of capacity increase
k_{13}	0.01	Dividend percentage
$k_{i,1}$	$(65, 35)$	Initial production capacity of P_i, $1 \leq i \leq k_1$
$k_{i,2}$	$(13, 13)$	Price fixed by P_i for each unit of d_i
$k_{j,3}$	$(50, 30, 20)$	Initial capacity of R_j, $1 \leq j \leq k_2$
$k_{j,6}$	$(15, 15, 15)$	Price fixed by R_j for each order of good j, $1 \leq j \leq k_2$

Table 2. Parameters utilized for simulation of enhanced model

the capacity increase mechanism. As we have seen in the previous chapters, during the evolution of the system, depreciation mechanism pushes capacity down and capacity increase mechanism competes with the previous one to maintain system evolution alive.

Case C: (capacity depreciation + capacity increase mechanism + dividend payment deactivated). In this case, we will show how the previous evolution changes when dividend payment mechanism is deactivated. Depreciation mechanism pushes capacity down, capacity increase mechanism competes with the previous one to maintain system evolution alive but all these processes are not possible is not possible if there are no enough movement of monetary units between all the actors in the system.

Once situation is restored to case B, the following stable behavior of u_C is obtained:

Clearly these three mechanisms cooperation (capacity depreciation, capacity increase decision and monetary unit flow mechanisms) is crucial to the stable

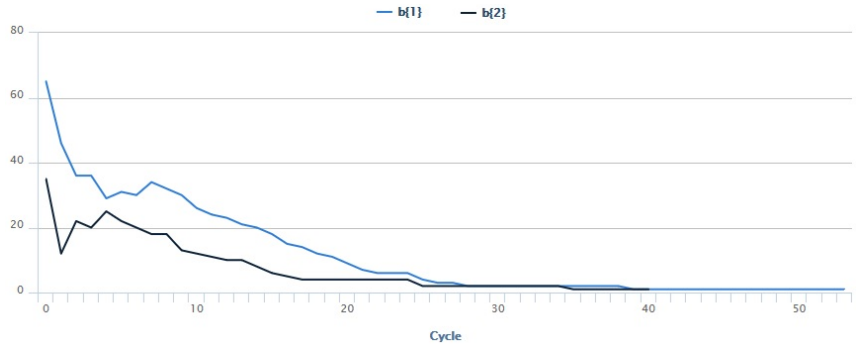


Fig. 5. Evolution of b_i in presence of depreciation standalone

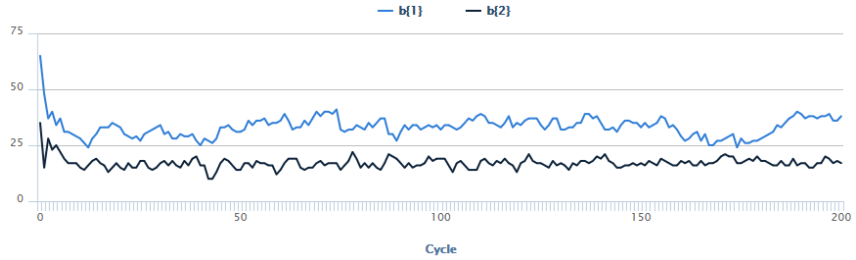


Fig. 6. Evolution of b_i in presence of depreciation and capacity increase mechanism

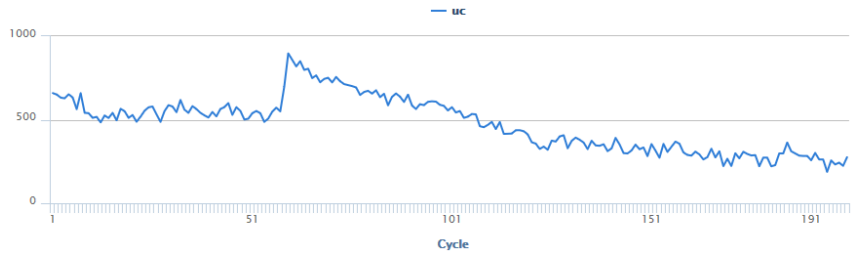


Fig. 7. Evolution of C monetary units with dividend payment mechanism deactivated

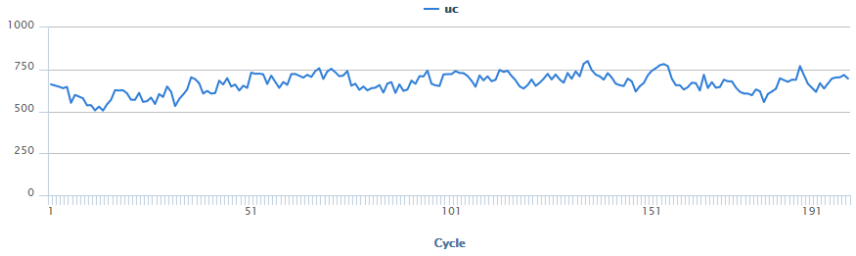


Fig. 8. Evolution of C monetary units with dividend payment mechanism activated

evolution of the system. Next figures show the evolution of the rest of variables of the system.

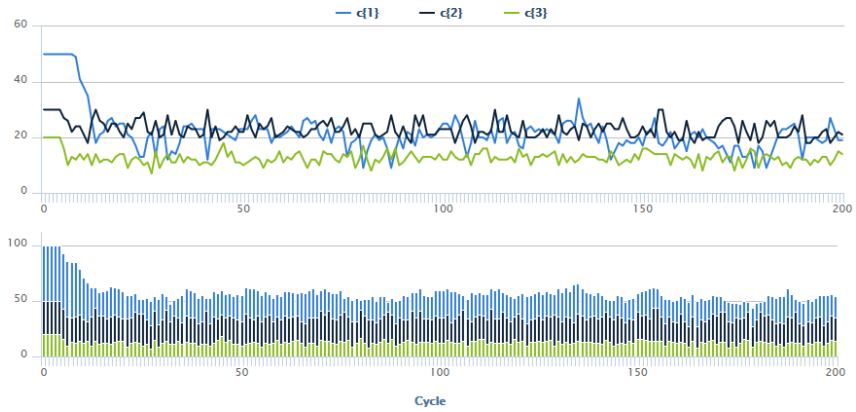


Fig. 9. Evolution of retailers capacities in line chart and accumulated columns

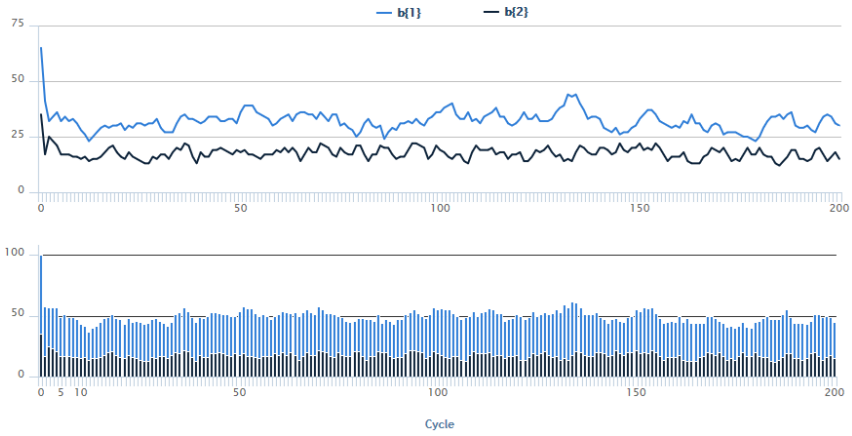


Fig. 10. Evolution of producers capacities in line chart and accumulated columns

5 Conclusions

In our work, we have implemented the ideas sketched by Gh. and R. Păun [10] into a specific P system framework (more specifically, PDP systems). The results

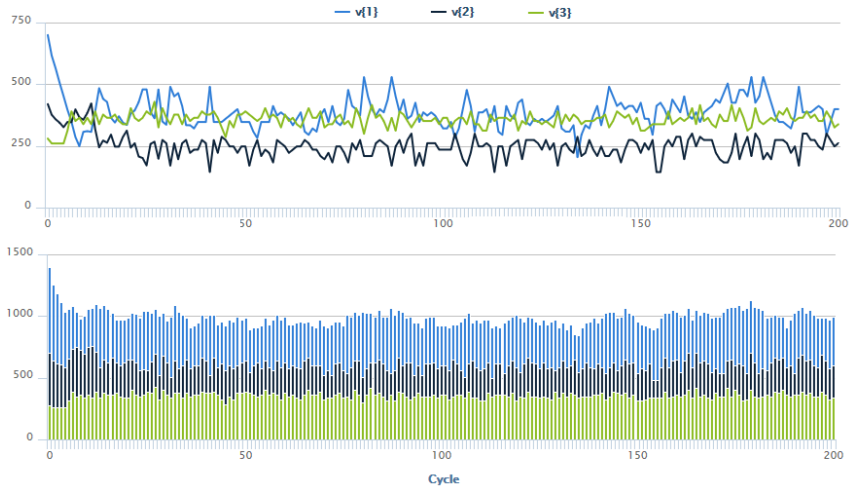


Fig. 11. Evolution of retailers monetary units in line chart and accumulated columns

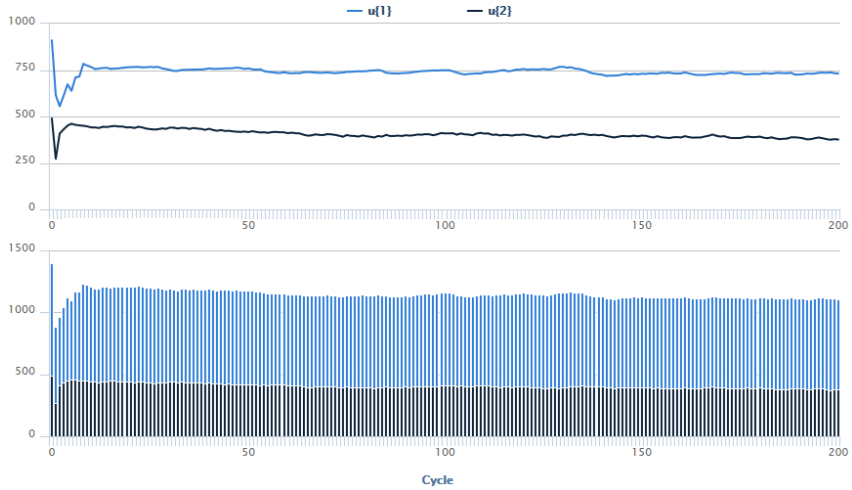


Fig. 12. Evolution of producers monetary units in line chart and accumulated columns

obtained look very promising. Firstly, we have been able to replicate Păuns' numerical results, designing a model for the producer-retailer problem and simulating it using `MeCoSim` [12]. In this initial model, some basic interactions between producers and retailers are considered, such as, production of goods from raw material, reception of orders from consumers and purchase transaction to match this goods and orders. It includes several simplifications and an exogenous artificial injection of monetary units that prevents system from reaching a halting configuration.

Aimed by these results we have proposed an enhanced model for the producer-retailer problem. The initial model has been enriched considering a plethora of phenomena to move it closer to the complexities of real world. We have taken advantage from **modularity**, one of the main membrane computing advantages for systems modelling. Hence, it has not been necessary to build again the model from scratch, we have added complexity to the model over the initial layer. Many real economic world interactions have been included in the model as a new layer: capital stock depreciation, capacity increase decision mechanism, costs of capital (rents for its owners), dividend payments, and a general idea of making monetary units flow across the system. Additionally, randomness has been introduced, in several steps of the model, by means of mechanisms frequently used in PDP world.

This enhanced model has also been simulated using `MeCoSim` and system evolution was analysed in depth. Some remarkable facts are that system can evolve autonomously without any exogenous influence. Although initial values of variables are settled, they change their values reaching an equilibrium point. Once reached this stability point, system varies slightly around it. From the previous results, we can derive that multiple economic issues can be modelled using membrane computing. Therefore, more efforts must be done in this direction.

Acknowledgements

This work was partially supported by Grant numbers 61472328 and 61320106005 of the National Natural Science Foundation of China.

References

1. J. Bartosik: Paun's systems in modeling of human resource management. *Second Conference on Tools and Methods of Data Transformation*, WSU Kielce, 2004.
2. M. Cardona, M.A. Colomer, M.J. Prez-Jimnez, Sanuy, D., Margalida, A., 2009. Modeling ecosystems using P Systems: The Bearded vulture, a case study. *Lecture Notes in Computer Science*, **5391** (2009), 137156.
3. S. Cheruku, A. Păun, F.J. Romero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems. *Progress in Natural Science*, **17**, 4 (2007), 424-431.
4. M.A. Colomer, S. Lavin, I. Marco, A. Margalida, I. Prez-Hurtado, M.J. Prez-Jimnez, D. Sanuy, E. Serrano, and L. Valencia-Cabrera. Modeling population growth of pyrenean chamois (rupicapra p. pyrenaica) by using p-systems. *Lecture Notes in Computer Science*, **6501** (2011), 144-159.
5. J.M. Epstein, R. Axtell. *Growing Artificial SocietiesSocial Science from the Bottom Up*. Brookings Institution Press and The MIT Press, 1996.
6. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A. An overview of P-Lingua 2.0. *Lecture Notes in Computer Science*, 5957 (2010), 264-288.

7. W. Korczynski. On a model of economic systems. *Second Conference on Tools and Methods of Data Transformation*, WSU Kielce, 2004.
8. W. Korczynski. Paun's systems and accounting. In *Pre-Proceedings of Sixth Workshop on Membrane Computing*, WMC6, Vienna, July 2005, 461-464.
9. Gh. Păun. *Membrane Computing: An introduction*. Springer, 2002.
10. Gh. Păun, R. Păun. Membrane computing as a framework for modeling economic processes. *Proceedings of SYNASC 05*, Timisoara, Romania, IEEE Press, 2005, 1118.
11. Gh. Păun, R. Păun. Membrane Computing and Economics. In Gh. Păun, G. Rozenberg, A. Salomaa, (eds.) *The Oxford Handbook of Membrane Computing*. Oxford University Press, New York, 2010, 632-644.
12. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez. Mecosim: A general purpose software tool for simulating biological phenomena by means of p systems. *IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010)*, I (2010), 637–643.

Restricted Polarizationless P Systems with Active Membranes: Minimal Cooperation Only Inwards

Luis Valencia-Cabrera, David Orellana-Martín, Miguel Á. Martínez-del-Amor, Agustín Riscos-Núñez, Mario J. Pérez-Jiménez

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

Summary. Membrane computing is a computing paradigm providing a class of distributed parallel computing devices of a biochemical type whose process units represent biological membranes. In the cell-like basic model, a hierarchical membrane structure formally described by a rooted tree is considered. It is well known that families of such systems where the number of membranes can only decrease during a computation (for instance by dissolving membranes), can only solve in polynomial time problems in class **P**. *P systems with active membranes* is a variant where membranes play a central role in their dynamics. In the seminal version, membranes have an electrical polarization (positive, negative, or neutral) associated in any instant, and besides being dissolved, they can also replicate by using division rules. These systems are computationally universal, that is, equivalent in power to deterministic Turing machines, and computationally efficient, that is, able to solve computationally hard problems in polynomial time. If polarizations in membranes are removed and dissolution rules are forbidden, then only problems in class **P** can be solved in polynomial time by these systems (even in the case when division rules for non-elementary membranes are permitted). In that framework it has been shown that by considering minimal cooperation (left-hand side of such rules consists of at most two symbols) and minimal production (only one object is produced by the application of such rules) in object evolution rules, such systems provide efficient solutions to **NP**–complete problems. In this paper, minimal cooperation and minimal production in communication rules instead of object evolution rules is studied, and the computational efficiency of these systems is obtained in the case where division rules for non-elementary membranes are permitted.

Key words: Membrane Computing, polarizationless P systems with active membranes, cooperative rules, the **P** versus **NP** problem, SAT problem.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing providing distributed parallel and non-deterministic computing models whose computational devices are called *membrane systems* having units processor called *compartments*. This computing paradigm is inspired by some basic biological features, by the structure and functioning of the living cells, as well as from the cooperation of cells in tissues, organs, and organisms. Celllike membrane systems use the biological membranes arranged hierarchically, inspired from the structure of the cell.

In Membrane Computing, some variants capture the fact that membranes are not at all passive from a biochemistry view, for instance, the passing of a chemical compound through a membrane is often done by a direct interaction with the membrane itself. Some variants of P systems where the central role in their dynamics is played by the membranes have been considered. In these models, the membranes not only directly mediate the evolution and the communication of objects, but they can replicate themselves by means of a division process. Inspired by these features, *P systems with active membranes* [6] were introduced, based on processing multisets by means of non-cooperative rewriting rules, that is, rules where its left-hand side has at most only one object. Specifically, objects evolve inside membranes which can communicate between each other, can dissolve, and moreover (inspired by cellular mitosis process) can replicate by means of division rules. It is assumed that each membrane has associated an electrical polarization in any instant, one of the three possible: positive, negative, or neutral.

P systems with active membranes are computationally complete, that is, any recursively enumerable set of vectors of natural numbers (in particular, each recursively enumerable set of natural numbers) can be generated by such a system [6]. Hence, they are equivalent in power to deterministic Turing machines.

What about the computational efficiency of P systems with active membranes? The key is certainly in the use of division rules, as we can deduce from the so-called *Milano theorem* [13]: *A deterministic P system with active membranes but without membrane division can be simulated by a deterministic Turing machine with a polynomial slowdown.*

However, P systems with active membranes which make use of division rules have the ability to provide efficient solutions to computationally hard problems, by making use of an exponential workspace created in a polynomial time. Specifically, **NP**-complete problems can be solved in polynomial time by families of P systems with active membranes, without dissolution rules and which use division rules only for elementary membranes [6]. Moreover, the class of decision problems which can be solved by families of P systems with active membranes with dissolution rules and which use division for elementary and non-elementary membranes is equal to **PSPACE** [8]. Consequently, the usual framework of P systems with active membranes and electrical polarizations for solving decision problems seems to be too powerful from the computational complexity point of view.

With respect to the computational efficiency, in the classical framework of P system with active membranes, dissolution rules play an "innocent" role as well as

division for non-elementary membranes. However, if electrical charges are removed then these kind of rules come to play a relevant role. Specifically, P systems with active membranes and without electrical charges were initially studied in [1, 2] by replacing electrical charges by the ability to change the label of the membranes. In this paper, polarizationless P systems with active membranes where labels of membranes keep unchanged by the application of rules, are considered. In this new framework, if dissolution rules are forbidden then only problems in class **P** can be solved in an efficient way, even in the case that division for non-elementary membranes are permitted [5]. Is the class of polarizationless P systems with active membranes, with dissolution but using only division rules for elementary membranes computationally efficient? If $\mathbf{P} \neq \mathbf{NP}$, which is at all expected, then it is an open question, so-called *Păun's conjecture*.

In the seminal paper where P systems with active membranes were introduced, Gh. Păun says that "*working with non-cooperative rules is natural from a mathematical point of view but from a biochemical point of view this is not only non-necessary, but also non-realistic: most of the chemical reactions involve two or more than two chemical compounds* (*and also produce two or more than two compounds*)". In this context, a restricted cooperation has been considered in the classical framework of polarizationless P systems with active membranes. Specifically, minimal cooperation (the left-hand side and the right-hand side of any rules have, at most, two objects) in object evolution rules, has been previously studied from a computational complexity point of view. A polynomial-time solution to the `SAT` problem by means of families of polarizationless P systems with active membranes, with minimal cooperation in object evolution rules, has been provided [9]. Recently, this result has been improved by considering minimal cooperation in object evolution rules with and additional restriction: the right-hand side of any rules has only one object (called *minimal cooperation and minimal production*) [11]. A relevant fact in these results is the following: dissolution rules and division rules for non-elementary membranes are not necessary to reach the computational efficiency.

In this paper the role of minimal cooperation and minimal production in communication rules instead of object evolution rules, is studied from a complexity point of view. Specifically, by using families of membrane systems which use these syntactical ingredients, a polynomial-time solution to the `SAT` problem is provided but allowing division rules for non-elementary membranes.

The paper is structured as follows. First, some basic notions are recalled and the terminology and notation to be used in the paper is presented. Then, Section 3 introduces the model that will be investigated in this paper: polarizationless P systems with active membranes, with minimal cooperation and minimal production in their communication rules. Section 4 contains the main result of this paper, showing that these systems are capable of solving an **NP**-complete problem in an *efficient* way. Finally, the paper concludes with some final remarks and ideas for future work.

2 Preliminaries

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

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

2.1 Graphs and trees

Let us recall some notions related with graph theory (see [3] for details). An *undirected graph* is an ordered pair (V, E) where V is a set whose elements are called nodes or vertices and $E = \{\{x, y\} \mid x \in V, y \in V, x \neq y\}$ whose elements are called *edges*. A *path* of length $k \geq 1$ from a node u to a node v in a graph (V, E) is a finite sequence $(x_0, x_1, \ldots, x_k)$ of nodes such that $x_0 = u$, $x_k = v$ and $\{x_i, x_{i+1}\} \in E$. If $k \geq 2$ and $x_0 = x_k$ then we say that the path is a *cycle* of the graph. A graph with no cycle is said to be *acyclic*. An undirected graph is *connected* if there exist paths between every pair of nodes.

A *rooted tree* is a a connected, acyclic, undirected graph in which one of the vertices (called *the root of the tree*) is distinguished from the others. Given a node x (different from the root), if the last edge on the (unique) path from the root of the tree to the node x is $\{x, y\}$ (in this case, $x \neq y$), then y is **the** *parent* of node x and x is **a** *child* of node y. The root is the only node in the tree with no parent. A node with no children is called a *leaf*.

2.2 The Cantor pairing function

The Cantor pairing function encodes pairs of natural numbers by single natural numbers, and it is defined as follows: for each $n, p \in \mathbb{N}$

$$\langle n, p \rangle = \frac{(n+p)(n+p+1)}{2} + n$$

The Cantor pairing function is a primitive recursive function and bijective from $\mathbb{N} \times \mathbb{N}$ onto $\mathbb{N}$. Then, for each $t \in \mathbb{N}$ there exist unique natural numbers $n, p \in \mathbb{N}$ such that $t = \langle n, p \rangle$.

2.3 Decision problems and languages

A decision problem X is an ordered pair (I_X, θ_X), where I_X is a language over a finite alphabet Σ_X and θ_X is a total Boolean function over I_X. The elements of I_X are called *instances* of the problem X. Each decision problem X has associated a language L_X over the alphabet Σ_X as follows: $L_X = \{u \in {\Sigma_X}^* \mid \theta_X(u) = 1\}$. Conversely, every language L over an alphabet Σ has associated a decision problem $X_L = (I_{X_L}, \theta_{X_L})$ as follows: $I_{X_L} = \Sigma^*$ and $\theta_{X_L}(u) = 1$ if and only if $u \in L$. Therefore, given a decision problem X we have $X_{L_X} = X$, and given a language L over an alphabet Σ we have $L_{X_L} = L$. Then, solving a decision problem can be expressed equivalently as the task of recognizing the language associated with it.

2.4 Recognizer membrane systems

Recognizer membrane systems were introduced in [7] and they provide a natural framework to solve decision problems. This kind of systems are characterized by the following features: (a) the working alphabet Γ has two distinguished objects `yes` and `no`; (b) there exists an input membrane and an input alphabet Σ strictly contained in Γ; (c) the initial contents of the membranes are multisets over $\Gamma \setminus \Sigma$; (d) all computations halt; and (e) for each computation, either object `yes` or object `no` (but not both) must have been released into the environment, and only at the last step of the computation.

Given a recognizer membrane system, Π, for each multiset m over the input alphabet Σ we denote by $\Pi + m$ the membrane system Π with input multiset m, that is in the initial configuration of that system, the multiset m is added to the initial content of the input membrane. Thus, in a recognizer membrane system, Π, there exists an initial configuration associated with each multiset $m \in M_f(\Sigma)$.

3 Minimal cooperation and minimal production in communication rules

Definition 1. *A polarizationless P system with active membranes, with simple object evolution rules, without dissolution, with division rules for elementary and non-elementary membranes, and which makes use of minimal cooperation and minimal production in send-in communication rules, is a tuple*

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

where:

- Γ *is a finite alphabet whose elements are called objects and contains two distinguished objects* `yes` *and* `no`*.*
- $\Sigma \subsetneq \Gamma$ *is the input alphabet.*

- *H is a finite alphabet such that $H \cap \Gamma = \emptyset$ whose elements are called labels.*
- *$q \geq 1$ is the degree of the system.*
- *μ is a labelled rooted tree (called membrane structure) consisting of q nodes injectively labelled by elements of H (the root of μ is labelled by r_μ).*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are multisets over $\Gamma \setminus \Sigma$.*
- *$\mathcal{R}$ is a finite set of rules, of the following forms:*
  - (a_0) $[\, a \to b \,]_h$, *where* $h \in H$, $a, b \in \Gamma$, $u \in M_f(\Gamma)$ (simple object evolution rules).
  - (b_0) $a\, b\, [\ \ ]_h \to [\, c \,]_h$, *where* $h \in H \setminus \{r_\mu\}$, $a, b, c \in \Gamma$ (send–in communication rules with minimal cooperation and minimal production).
  - (c_0) $[\, a \,]_h \to b\, [\ \ ]_h$, *where* $h \in H$, $a, b \in \Gamma$ (send–out communication rules).
  - (d_0) $[\, a \,]_h \to b$, *where* $h \in H \setminus \{i_{out}, r_\mu\}$, $a, b \in \Gamma$ (dissolution rules).
  - (e_0) $[\, a \,]_h \to [\, b \,]_h\, [\, c \,]_h$, *where* $h \in H \setminus \{i_{out}, r_\mu\}$, $a, b, c \in \Gamma$ *and* h *is the label of an elementary membrane* μ (division rules for elementary membranes).
  - (f_0) $[\, [\ ]_{h_1} [\ ]_{h_2} \,]_{h_0} \to [\, [\ ]_{h_1} \,]_{h_0}\, [\, [\ ]_{h_2} \,]_{h_0}$, *where* $h_0, h_1, h_2 \in H$ *and* $h_0 \neq r_\mu$ (division rules for non-elementary membranes).
- *$i_{in} \in H$, $i_{out} \in H \cup \{env\}$ (if $i_{out} \in H$ then i_{out} is the label of a leaf of μ).*

In a similar way is defined the concept of "*polarizationless P system with active membranes, with simple object evolution rules, without dissolution, with division rules for elementary and non-elementary membranes, and which makes use of minimal cooperation and minimal production in send-out communication rules*". The only difference concerns rules of type (b_0) and (c_0). In this case are, respectively:

(b_0') $a\, [\ \ ]_h \to [\, b \,]_h$ for $h \in H \setminus \{r_\mu\}$, $a, b \in \Gamma$ *(send–in communication rules).*
(c_0') $[\, a\, b \,]_h \to c\, [\ \ ]_h$ for $h \in H$, $a, b, c \in \Gamma$ *(send–out communication rules with minimal cooperation and minimal production).*

The semantics of this kind of P systems follows the usual principles of P systems with active membranes [6].
We denote by $\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)$ (respectively, $\mathcal{DAM}^0(+e_s, mcmp_{out}, -d, +n)$) the class of all recognizer polarizationless P system with active membranes, with simple object evolution rules, without dissolution, with division rules for elementary and non-elementary membranes, which make use of minimal cooperation and minimal production in send-in (respectively, send-out) communication rules.

4 Solving SAT in $\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)$

In this section, a polynomial-time solution to SAT problem, is explicitly given in the framework of recognizer polarizationless P systems with active membranes with simple object evolution rules, without dissolution and with division rules for

elementary and non-elementary membranes which make use of minimal cooperation and minimal production in send-in communication rules. For that, a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of recognizer P systems from $\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)$ will be presented.

4.1 Description of a solution to SAT problem in $\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)$

For each $n, p \in \mathbb{N}$, we consider the recognizer P system

$$\Pi(\langle n, p\rangle) = (\Gamma, \Sigma, H, \mu, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in}, i_{out})$$

from $\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)$, defined as follows:

(1) Working alphabet:

$$\begin{aligned}\Gamma = \Sigma \cup \{\texttt{yes}, \texttt{no}, \#\} \cup \{a_{i,k} \mid 1 \le i \le n \wedge 1 \le k \le 2i-1\} \cup \\ \{\alpha_k \mid 0 \le k \le 4np+2n+2p+1\} \cup \{\beta_k \mid 0 \le k \le 4np+2n+2p+2\} \cup \\ \{\gamma_k \mid 0 \le k \le 4np+2n\} \cup \\ \{t_{i,k}, f_{i,k} \mid 1 \le i \le n \wedge 2i-1 \le k \le 2n+2p-1\} \cup \{T_i, F_i \mid 1 \le i \le n\} \cup \\ \{c_j \mid 1 \le j \le p\} \cup \{c_{j,k} \mid 1 \le j \le p \wedge 0 \le k \le np-1\} \cup \\ \{d_j \mid 1 \le j \le p\} \cup \{x_{i,j,k}, \overline{x}_{i,j,k}, x^*_{i,j,k} \mid 1 \le i \le n \wedge 1 \le j \le p \wedge \\ 1 \le k \le 2n+2np+n(j-1)+(i-1)\}\end{aligned}$$

where the input alphabet is $\Sigma = \{x_{i,j,0}, \overline{x}_{i,j,0}, x^*_{i,j,0} \mid 1 \le i \le n \wedge 1 \le j \le p\}$;

(2) $H = \{0, 1, 2\}$;

(3) Membrane structure: $\mu = [\,[\,[\,\,]_2\,]_1\,]_0$, that is, $\mu = (V, E)$ where $V = \{0, 1, 2\}$ and
$E = \{(0, 1), (1, 2)\}$;

(4) Initial multisets:
$\mathcal{M}_0 = \{\alpha_0, \beta_0\}$, $\mathcal{M}_1 = \{\gamma_0\} \cup \{T_i^p, F_i^p \mid 1 \le i \le n\}$, $\mathcal{M}_2 = \{a_{i,1} \mid 1 \le i \le n\}$;

(5) The set of rules $\mathcal{R}$ consists of the following rules:

5.1 Counters for synchronize the answer of the system.

$$\begin{aligned}&[\,\alpha_k \longrightarrow \alpha_{k+1}\,]_0, \text{ for } 0 \le k \le 4np+2n+2p \\ &[\,\beta_k \longrightarrow \beta_{k+1}\,]_0, \text{ for } 0 \le k \le 4np+2n+2p+1 \\ &[\,\gamma_k \longrightarrow \gamma_{k+1}\,]_1, \text{ for } 0 \le k \le 4np+2n-1\end{aligned}$$

5.2 Rules to generate 2^n membranes labelled by 1 and 2^n membranes labelled by 2 (these encoding all possible truth assignment of n variables of the input formula).

$$[\,a_{i,2i-1}\,]_2 \longrightarrow [\,t_{i,i}\,]_2\,[\,f_{i,i}\,]_2, \text{ for } 1 \le i \le n$$

$$\begin{aligned}&[\,a_{i,j} \longrightarrow a_{i,j+1}\,]_2, \text{ for } 2 \le i \le n, 1 \le j \le 2i-2 \\ &[\,[\,\,]_2\,[\,\,]_2\,]_1 \longrightarrow [\,[\,\,]_2\,]_1\,\,[\,[\,\,]_2\,]_1\end{aligned}$$

$$\left.\begin{aligned}&[\,t_{i,j} \longrightarrow t_{i,j+1}\,]_2 \\ &[\,f_{i,j} \longrightarrow f_{i,j+1}\,]_2\end{aligned}\right\}, \text{ for } 1 \le i \le n, i \le j \le 2n-1$$

5.3 Rules to produce exactly p copies of each truth assignment encoded by membranes labelled by 2.

$$\left.\begin{array}{l}[t_{i,2jn}]_2 \longrightarrow t_{i,2jn+1}\,[\,]_2 \\ [f_{i,2jn}]_2 \longrightarrow f_{i,2jn+1}\,[\,]_2\end{array}\right\},\ \text{for } 1 \le i \le n,\ 1 \le j \le p$$

$$\left.\begin{array}{l}[\,t_{i,2jn+k} \longrightarrow t_{i,2jn+k+1}\,]_1 \\ [\,f_{i,2jn+k} \longrightarrow f_{i,2jn+k+1}\,]_1\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p,\\ 1 \le k \le n-1\end{array}$$

$$\left.\begin{array}{l}t_{i,(2j+1)n}\ F_i[\ \ ]_2 \longrightarrow [\,t_{i,(2j+1)n+1}\,]_2 \\ f_{i,(2j+1)n}\ T_i[\ \ ]_2 \longrightarrow [\,f_{i,(2j+1)n+1}\,]_2\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p-1\end{array}$$

$$\left.\begin{array}{l}[\,t_{i,(2j+1)n+k} \longrightarrow t_{i,(2j+1)n+k+1}\,]_2 \\ [\,f_{i,(2j+1)n+k} \longrightarrow f_{i,(2j+1)n+k+1}\,]_2\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p\end{array}$$

$$\left.\begin{array}{l}t_{i,2np+n}\ F_i[\ \ ]_2 \longrightarrow [\#]_2 \\ f_{i,2np+n}\ T_i[\ \ ]_2 \longrightarrow [\#]_2\end{array}\right\},\ \text{for } 1 \le i \le n$$

5.4 Rules to prepare the input formula for check clauses:

$$\left.\begin{array}{l}[\,x_{i,j,k} \longrightarrow x_{i,j,k+1}\,]_1 \\ [\,\overline{x}_{i,j,k} \longrightarrow \overline{x}_{i,j,k+1}\,]_1 \\ [\,x^*_{i,j,k} \longrightarrow x^*_{i,j,k+1}\,]_1\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p,\\ 0 \le k \le 2np+2n\\ +n(j-1)+(i-1)-1\end{array}$$

5.5 Rules implementing the first checking stage.

$$\left.\begin{array}{l}T_i\ x_{i,j,2np+2n+n(j-1)+(i-1)}[\ \ ]_2 \longrightarrow [c_{j,0}]_2 \\ T_i\ \overline{x}_{i,j,2np+2n+n(j-1)+(i-1)}[\ \ ]_2 \longrightarrow [\#]_2 \\ T_i\ x^*_{i,j,2np+2n+n(j-1)+(i-1)}[\ \ ]_2 \longrightarrow [\#]_2 \\ F_i\ x_{i,j,2np+2n+n(j-1)+(i-1)}[\ \ ]_2 \longrightarrow [\#]_2 \\ F_i\ \overline{x}_{i,j,2np+2n+n(j-1)+(i-1)}[\ \ ]_2 \longrightarrow [c_{j,0}]_2 \\ F_i\ x^*_{i,j,2np+2n+n(j-1)+(i-1)}[\ \ ]_2 \longrightarrow [\#]_2\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p\end{array}$$

5.6 Rules implementing the second checking stage.

$$[c_{j,k} \longrightarrow c_{j,k+1}]_2,\ \text{for } 1 \le j \le p,\ 0 \le k \le np-2$$
$$[c_{j,np-1}]_2 \longrightarrow c_j[\ \ ]_2,\ \text{for } 1 \le j \le p$$
$$\gamma_{4np+2n}\ c_1[\ \ ]_2 \longrightarrow [\,d_1\,]_2$$
$$[\,d_j\,]_2 \longrightarrow d_j\,[\ \ ]_2,\ \text{for } 1 \le j \le p$$
$$d_j\ c_{j+1}\,[\ \ ]_2 \longrightarrow [\,d_{j+1}\,]_2,\ \text{for } 1 \le j \le p-1$$

5.7 Rules to provide the correct answer of the system.

$$[\,d_p\,]_1 \longrightarrow d_p[\ \ ]_1$$
$$\alpha_{4np+2n+2p+1}\ d_p[\ \ ]_1 \longrightarrow [\,\texttt{yes}\,]_1$$
$$\alpha_{4np+2n+2p+1}\ \beta_{4np+2n+2p+2}[\ \ ]_1 \longrightarrow [\,\texttt{no}\,]_1$$
$$[\,\texttt{yes}\,]_1 \longrightarrow \texttt{yes}[\ \ ]_1$$
$$[\,\texttt{no}\,]_1 \longrightarrow \texttt{no}[\ \ ]_1$$
$$[\,\texttt{yes}\,]_0 \longrightarrow \texttt{yes}[\ \ ]_0$$
$$[\,\texttt{no}\,]_0 \longrightarrow \texttt{no}[\ \ ]_0$$

(6) the input membrane is the membrane labelled by 1 ($i_{in} = 1$) and the output region is the environment ($i_{out} = env$).

5 A formal verification

Let $\varphi = C_1 \wedge \ldots \wedge C_p$ an instance of SAT problem consisting of p clauses $C_j = l_{j,1} \vee \ldots \vee l_{j,r_j}$, $1 \leq j \leq p$, where $Var(\varphi) = \{x_1, \ldots, x_n\}$, and $l_{j,k} \in \{x_i, \neg x_i | 1 \leq i \leq n\}$, $1 \leq j \leq p$, $1 \leq k \leq r_j$. Let us asume that the number of variables, n, and the number of clauses, p, of φ, are greater or equal to 2.

We consider the polynomial encoding (cod, s) from SAT in $\mathbf{\Pi}$ defined as follows: For each $\varphi \in I_{\mathtt{SAT}}$ with n variables and p clauses, $s(\varphi) = \langle n, p \rangle$ and

$$cod(\varphi) = \{x_{i,j,0} | x_i \in C_j\} \cup \{\overline{x}_{i,j,0} | \neg x_i \in C_j\} \cup \{x^*_{i,j,0} | x_i \notin C_j, \neg x_i \notin C_j\}$$

For instance, the formula $\varphi = (x_1 + x_2 + \overline{x}_3)(\overline{x}_2 + x_4)(\overline{x}_2 + x_3 + \overline{x}_4)$ is encoded as follows:

$$cod(\varphi) = \begin{pmatrix} x_{1,1,0} & x_{2,1,0} & \overline{x}_{3,1,0} & x^*_{4,1,0} \\ x^*_{1,2,0} & \overline{x}_{2,2,0} & x^*_{3,2,0} & x_{4,2,0} \\ x^*_{1,3,0} & \overline{x}_{2,3,0} & x_{3,3,0} & \overline{x}_{4,3,0} \end{pmatrix}$$

That is, j-th row ($1 \leq j \leq p$) represents the j-th clause C_j of φ. We denote $(cod(\varphi))^p_j$ the code of the clauses $C_j, \ldots, C_p$, that is, the expression containing from j-th row to p-th row. For instance,

$$cod(\varphi)^p_2 = \begin{pmatrix} x^*_{1,2,0} & \overline{x}_{2,2,0} & x^*_{3,2,0} & x_{4,2,0} \\ x^*_{1,3,0} & \overline{x}_{2,3,0} & x_{3,3,0} & \overline{x}_{4,3,0} \end{pmatrix}$$

We denote $(cod_k(\varphi))^p_j)$ the code $cod(\varphi)^p_j$ when the third index of the variables equal 3. For instance: row to p-th row. For instance,

$$cod_3(\varphi)^p_2 = \begin{pmatrix} x^*_{1,2,3} & \overline{x}_{2,2,3} & x^*_{3,2,3} & x_{4,2,3} \\ x^*_{1,3,3} & \overline{x}_{2,3,3} & x_{3,3,3} & \overline{x}_{4,3,3} \end{pmatrix}$$

We denote $(cod'_k(\varphi))^p_j)$ the code $cod(\varphi)^p_j$ when the third index of the variables equal 3. For instance: row to p-th row. For instance,

$$cod'_3(\varphi)^p_2 = \begin{pmatrix} {x^*}'_{1,2,3} & \overline{x}'_{2,2,3} & {x^*}'_{3,2,3} & x'_{4,2,3} \\ {x^*}'_{1,3,3} & \overline{x}'_{2,3,3} & x'_{3,3,3} & \overline{x}'_{4,3,3} \end{pmatrix}$$

We denote $(cod^*(\varphi))^p_j)$ the code $cod(\varphi)^p_j$ when the third index does not exist. For instance: row to p-th row. For instance,

$$cod^*(\varphi)^p_2 = \begin{pmatrix} {x^*}_{1,2} & \overline{x}_{2,2} & x^*_{3,2} & x_{4,2} \\ {x^*}_{1,3} & \overline{x}_{2,3} & x_{3,3} & \overline{x}_{4,3} \end{pmatrix}$$

The Boolean formula φ will be processed by the system $\Pi(s(\varphi)) + cod(\varphi)$. Next, we informally describe how that system works.

The solution proposed follows a brute force algorithm in the framework of recognizer P systems with active membranes, minimal cooperation in object evolution rules and division rules only for elementary membranes, and it consists of the following stages:

- *Generation stage*: using separation rules, beside other rules that make a "simulation" of division rules, we get all truth assignments for the variables $\{x_1, \ldots, x_n\}$ associated with φ are produced. Specifically, 2^n membranes labelled by 2 and 2^n labelled by 1 are generated. Each of the former ones encodes a truth assignment. This stage takes exactly $2n + 2np$ steps, being n the number of variables of φ.
- *First Checking stage*: checking whether or not each clause of the input formula φ is satisfied by the truth assignments generated in the previous stage, encoded by each membrane labelled by 2. This stage takes exactly np steps, being n the number of the variables and p the number of clauses of φ.
- *Second Checking stage*: checking whether or not all clauses of the input formula φ are satisfied by some truth assignment encoded by a membrane labelled by 2. This stage takes exactly $np + 2p$ steps, being n the number of variables and p the number of clauses of φ.
- *Output stage*: the system sends to the environment the right answer according to the results of the previous stage. This stage takes 4 steps if the answer is `yes` and 5 steps if the answer is `no`.

5.1 Generation stage

Through this stage, all the different truth assignments for the variables associated with the Boolean formula φ will be generated within membranes labelled by 1, by the applications of rules from **5.2** and **5.3**. In the first $2n$ steps, 2^n membranes labelled by 2 and 2^n membranes labelled by 1, alternating between the division of membranes labelled by 2 (in odd steps) and the division of membranes labelled by 1 (in even steps).

Proposition 1. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a_0) *For each $2k$ ($0 \leq k \leq n-1$) at configuration $\mathcal{C}_{2k}$ we have the following:*
- $\mathcal{C}_{2k}(0) = \{\alpha_{2k}, \beta_{2k}\}$
- *There are 2^k membranes labelled by 1 such that each of them contains*
  - ⋆ *the input multiset $cod_{2k}(\varphi)$;*
  - ⋆ *an object γ_{2k}; and*
  - ⋆ *p copies of every T_i and F_i, $1 \leq i \leq n$.*
- *There are 2^k membranes labelled by 2 such that each of them contains*
  - ⋆ *objects $a_{i,2k+1}$, $k+1 \leq i \leq n$; and*
  - ⋆ *a different subset $\{r_{1,j}, \ldots, r_{k,j}\}$, $k+1 \leq j \leq 2k$, being $r \in \{t, f\}$.*

(a_1) *For each $2k+1$ ($0 \leq j \leq n-1$) at configuration $\mathcal{C}_{2k+1}$ we have the following:*
- $\mathcal{C}_{2k+1}(0) = \{\alpha_{2k+1}, \beta_{2k+1}\}$
- *There are 2^k membranes labelled by 1 such that each of them contains*
  - ⋆ *the input multiset $cod_{2k+1}(\varphi)$;*
  - ⋆ *an object γ_{2k+1}; and*
  - ⋆ *p copies of every T_i and F_i, $1 \leq i \leq n$.*

- *There are* 2^{k+1} *membranes labelled by 2 such that each of them contains*
  - ⋆ *objects* $a_{i,2(k+1)}$, $k+1 \leq i \leq n$; *and*
  - ⋆ *a different subset* $\{r_{1,j}, \ldots, r_{k+1,j}\}$, $k+1 \leq j \leq 2k+1$, *being* $r \in \{t, f\}$.

(*b*) $\mathcal{C}_{2n}(0) = \{\alpha_{2n}, \beta_{2n}\}$, *and in* $\mathcal{C}_{2n}$ *there are* 2^n *membranes labelled by 1, such that each of them contains the input multiset* $cod_{2n}(\varphi)$, *p copies of every* T_i *and* F_i *(*$1 \leq i \leq n$*) and an object* γ_{2n}; *and* 2^n *membranes labelled by 2, such that each of them contains a different subset of objects* $r_{i,2n+1-i}$, $1 \leq i \leq n$

Proof. (*a*) is going to be demonstrated by induction on k

- The base case $k = 0$ is trivial because:

  (a_0) at the initial configuration $\mathcal{C}_0$ we have: $\mathcal{C}_0(0) = \{\alpha_0, \beta_0\}$ and there exists a single membrane labelled by 1 containing the input multiset $cod(\varphi)$, an object γ_0 and p copies of T_i and F_i, being $1 \leq i \leq n$; and a single membrane labelled by 2 containing the objects $a_{1,1}, \ldots, a_{n,1}$. Then, configuration $\mathcal{C}_0$ yields configuration $\mathcal{C}_1$ by applying the rules:

  $[\ a_{1,1}\ ]_2 \to [\ t_{1,1}\ ]_2\ [\ f_{1,1}\ ]_2$
  $[\ a_{i,1} \to a_{i,2}\ ]_2$, for $k+1 \leq i \leq n$
  $[\ \alpha_0 \to \alpha_1\ ]_0$
  $[\ \beta_0 \to \beta_1\ ]_0$
  $[\ \gamma_0 \to \gamma_1\ ]_1$
  $\left.\begin{array}{l} [\ x_{i,j,0} \to x_{i,j,1}\ ]_1 \\ [\ \overline{x}_{i,j,0} \to \overline{x}_{i,j,1}\ ]_1 \\ [\ x^*_{i,j,0} \to x^*_{i,j,1}\ ]_1 \end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

  (a_1) at $\mathcal{C}_1$ we have $\mathcal{C}_1(0) = \{\alpha_1, \beta_1\}$ and there exists a single membrane labelled by 1 containing the input multiset $cod_1(\varphi)$, an object γ_1 and p copies of T_i and F_i, being $1 \leq i \leq n$; and two membranes labelled by 2 containing the objects $a_{2,2}, \ldots, a_{n,2}$ and one with the object $t_{1,1}$ and the other one with the object $f_{1,1}$. Then, the configuration $\mathcal{C}_1$ yields configuration $\mathcal{C}_2$ by applying the rules:

  $[\ t_{1,1} \to t_{1,2}\ ]_2$
  $[\ f_{1,1} \to f_{1,2}\ ]_2$
  $[\ [\ \ ]_2\ [\ \ ]_2]_1 \to [\ [\ \ ]_2]_1\ [\ [\ \ ]_2]_1$
  $[a_{i,2} \to a_{i,3}\ ]_2$, for $2 \leq i \leq n$
  $[\ \alpha_1 \to \alpha_2\ ]_0$
  $[\ \beta_1 \to \beta_2\ ]_0$
  $[\ \gamma_1 \to \gamma_2\ ]_1$
  $\left.\begin{array}{l} [\ x_{i,j,1} \to x_{i,j,2}\ ]_1 \\ [\ \overline{x}_{i,j,1} \to \overline{x}_{i,j,2}\ ]_1 \\ [\ x^*_{i,j,1} \to x^*_{i,j,2}\ ]_1 \end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

  Thus, $\mathcal{C}_2(0) = \{\alpha_2, \beta_2\}$, and there exist two membranes labelled by 1 containing the input multiset $cod_2(\varphi)$, an object γ_2 and p copies of T_i and F_i, being $1 \leq i \leq n$; and two membranes labelled by 2 containing the objects $a_{2,3}, \ldots, a_{n,3}$ and one with the object $t_{1,2}$ and the other one with the object $f_{1,2}$. Hence, the result holds for $k = 1$.

- Supposing, by induction, result is true for k $(0 \leq k \leq n-1)$
  - $\mathcal{C}_{2k}(0) = \{\alpha_{2k}, \beta_{2k}\}$
  - In $\mathcal{C}_{2k}$ there are 2^k membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{2k}(\varphi)$;
    - ⋆ an object γ_{2k}; and
    - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
  - In $\mathcal{C}_{2k}$ there are 2^k membranes labelled by 2 such that each of them contains
    - ⋆ objects $a_{i,2k+1}$, $k+1 \leq i \leq n$; and
    - ⋆ a different subset $\{r_{1,j}, \ldots, r_{k,j}\}$, $k+1 \leq j \leq 2k$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{2k}$ yields configuration $\mathcal{C}_{2k+1}$ by applying the rules:

    $[\ a_{k,2k+1}\ ]_2 \to [\ t_{k,k}\ ]_2\ [\ f_{k,k}\ ]_2$

    $[\ a_{i,2k+1} \to a_{i,2k+2}\ ]_2$, for $k+1 \leq i \leq n$

    $\left.\begin{array}{l}[\ t_{i,j} \to t_{i,j+1}\ ]_2 \\ [\ f_{i,j} \to f_{i,j+1}\ ]_2\end{array}\right\}$ for $1 \leq i \leq k-1, k+1 \leq j \leq 2k$

    $[\ \alpha_{2k} \to \alpha_{2k+1}\ ]_0$

    $[\ \beta_{2k} \to \beta_{2k+1}\ ]_0$

    $[\ \gamma_{2k} \to \gamma_{2k+1}\ ]_1$

    $\left.\begin{array}{l}[\ x_{i,j,2k} \to x_{i,j,2k+1}\ ]_1 \\ [\ \overline{x}_{i,j,1} \to \overline{x}_{i,j,2k+1}\ ]_1 \\ [\ x^*_{i,j,1} \to x^*_{i,j,2k+1}\ ]_1\end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

    Therefore, the following holds
  - $\mathcal{C}_{2k+1}(0) = \{\alpha_{2k+1}, \beta_{2k+1}\}$
  - In $\mathcal{C}_{2k+1}$ there are 2^k membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{2k+1}(\varphi)$;
    - ⋆ an object γ_{2k+1}; and
    - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
  - In $\mathcal{C}_{2k+1}$ there are 2^{k+1} membranes labelled by 2 such that each of them contains
    - ⋆ objects $a_{i,2(k+1)}$, $k+1 \leq i \leq n$; and
    - ⋆ a different subset $\{r_{1,j}, \ldots, r_{k+1,j}\}$, $k+1 \leq j \leq 2k+1$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{2k+1}$ yields configuration $\mathcal{C}_{2(k+1)}$ by applying the rules:

    $\left.\begin{array}{l}[\ t_{i,j} \to t_{i,j+1}\ ]_2 \\ [\ f_{i,j} \to f_{i,j+1}\ ]_2\end{array}\right\}$ for $1 \leq i \leq k+1, k+1 \leq j \leq 2k+1$

    $[\ [\ \ ]_2\ [\ \ ]_2]_1 \to [\ [\ \ ]_2]_1\ [\ [\ \ ]_2]_1$

    $[\ a_{i,2(k+1)} \to a_{i,2(k+1)+1}\ ]_2$, for $k+1 \leq i \leq n$

    $[\ \alpha_{2k+1} \to \alpha_{2(k+1)}\ ]_0$

    $[\ \beta_{2k+1} \to \beta_{2(k+1)}\ ]_0$

    $[\ \gamma_{2k+1} \to \gamma_{2(k+1)}\ ]_1$

    $\left.\begin{array}{l}[\ x_{i,j,2k+1} \to x_{i,j,2k+2}\ ]_1 \\ [\ \overline{x}_{i,j,2k+1} \to \overline{x}_{i,j,2k+2}\ ]_1 \\ [\ x^*_{i,j,2k+1} \to x^*_{i,j,2k+2}\ ]_1\end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

    Therefore, the following holds

- $\mathcal{C}_{2(k+1)}(0) = \{\alpha_{2(k+1)}, \beta_{2(k+1)}\}$
- In $\mathcal{C}_{2(k+1)}$ there are 2^{k+1} membranes labelled by 1 such that each of them contains
  - ⋆ the input multiset $cod_{2(k+1)}(\varphi)$;
  - ⋆ an object $\gamma_{2(k+1)}$; and
  - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
- In $\mathcal{C}_{2(k+1)}$ there are 2^{k+1} membranes labelled by 2 such that each of them contains
  - ⋆ objects $a_{i,2(k+1)+1}$, $k+1 \leq i \leq n$; and
  - ⋆ a different subset $\{r_{1,j}, \ldots, r_{k+1,j}\}$, $k+1 \leq j \leq 2(k+1)+1$.

  Hence, the result holds for $k+1$.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{2n-1}$ holds:
  - $\mathcal{C}_{2n-1}(0) = \{\alpha_{2n-1}, \beta_{2n-1}\}$
  - In $\mathcal{C}_{2n-1}$ there are 2^{n-1} membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{2n-1}p(\varphi)$;
    - ⋆ an object γ_{2n-1}; and
    - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
  - In $\mathcal{C}_{2n-1}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,2n-i}$, $1 \leq i \leq n$.

    Then, configuration $\mathcal{C}_{2n-1}$ yields $\mathcal{C}_{2n}$ by applying the rules:

$$\left.\begin{array}{l}[\ t_{i,2n-i} \to t_{i,2n+1-i}\ ]_2 \\ [\ f_{i,2n-i} \to f_{i,2n+1-1}\ ]_2\end{array}\right\} \text{ for } 1 \leq i \leq n$$

$$[\ [\ \ ]_2\ [\ \ ]_2]_1 \to [\ [\ \ ]_2]_1\ [\ [\ \ ]_2]_1$$

$$[\ \alpha_{2n-1} \to \alpha_{2n}\ ]_0$$

$$[\ \beta_{2n-1} \to \beta_{2n}\ ]_0$$

$$[\ \gamma_{2n-1} \to \gamma_{2n}\ ]_1$$

$$\left.\begin{array}{l}[\ x_{i,j,2n-1} \to x_{i,j,2n}\ ]_1 \\ [\ \overline{x}_{i,j,2n-1} \to \overline{x}_{i,j,2n}\ ]_1 \\ [\ x^*_{i,j,2n-1} \to x^*_{i,j,2n}\ ]_1\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p$$

    Then, we have $\mathcal{C}_{2n}(0) = \{\alpha_{2n}, \beta_{2n}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{2n}(\varphi)$, an object γ_{2n} and p copies of T_i and F_i, being $1 \leq i \leq n$; and 2^n membranes labelled by 2 containing a different multiset of objects $r_{i,2n+1-i}$, being $1 \leq i \leq n$.

□

When the tree structure is created, we start assigning a truth assignment to each branch. It is executed in the next $2np$ steps. The last n steps are different from the previous ones, so they deserve another proposition of the following one.

Proposition 2. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a_0) *For each* k $(1 \leq k \leq n)$ *and* l $(0 \leq l \leq p-1)$ *at configuration* $\mathcal{C}_{2n+2ln+k}$ *we have the following:*

- $\mathcal{C}_{2n+2ln+k}(0) = \{\alpha_{2n+2ln+k}, \beta_{2n+2ln+k}\}$
- *There are* 2^n *membranes labelled by 1 such that each of them contains*
  - ⋆ *the input multiset* $cod_{2n+2ln+k}(\varphi)$*;*
  - ⋆ *an object* $\gamma_{2n+2ln+k}$*;*
  - ⋆ p *copies of every* T_i *and* F_i*,* $1 \leq i \leq n$ *if the truth assignment associated to the branch contains its corresponding* t_i *or* f_i *object, and* $p-l$ *copies otherwise; and*
  - ⋆ *objects* $r_{i,2n+2ln+k-i+1}$*,* $1 \leq i \leq k$*, being* $r \in \{t, f\}$*.*
- *There are* 2^n *membranes labelled by 2 such that each of them contains a different subset of objects* $r_{i,2n+2ln+k-i+1}$*,* $k+1 \leq i \leq n$*, being* $r \in \{t, f\}$*.*

(a_1) *For each* k $(1 \leq k \leq n)$ *and* l $(0 \leq l \leq p-2)$ *at configuration* $\mathcal{C}_{3n+2ln+k}$ *we have the following:*

- $\mathcal{C}_{3n+2ln+k}(0) = \{\alpha_{3n+2ln+k}, \beta_{3n+2ln+k}\}$
- *There are* 2^n *membranes labelled by 1 such that each of them contains*
  - ⋆ *the input multiset* $cod_{3n+2ln+k}(\varphi)$*;*
  - ⋆ *an object* $\gamma_{3n+2ln+k}$*;*
  - ⋆ p *copies of every* T_i *and* F_i*,* $1 \leq i \leq n$ *if the truth assignment associated to the branch contains its corresponding* t_i *or* f_i *object; otherwise, there are* $p-l$ *objects if* $k+1 \leq i \leq n$*,* $p-l-1$ *otherwise; and*
  - ⋆ *objects* $r_{i,3n+2ln+k-i+1}$*,* $k+1 \leq i \leq n$*, being* $r \in \{t, f\}$*.*
- *There are* 2^n *membranes labelled by 2 such that each of them contains a different subset of objects* $r_{i,3n+2ln+k-i+1}$*,* $1 \leq i \leq k$*, being* $r \in \{t, f\}$*.*

(b) $\mathcal{C}_{n+2np}(0) = \{\alpha_{n+2np}, \beta_{n+2np}\}$*, and in* $\mathcal{C}_{n+2np}$ *there are* 2^n *membranes labelled by 1, such that each of them contains the input multiset* $cod_{n+2np}(\varphi)$*, an object* γ_{n+2np}*,* p *copies of every* T_i *and* F_i*,* $1 \leq i \leq n$ *if the truth assignment associated to the branch contains its corresponding* t_i *or* f_i *object, and 1 object otherwise and objects* $r_{i,n+2np-i+1}$*,* $1 \leq i \leq n$*, being* $r \in \{t, f\}$*, that is, the truth assignment associated with the branch; and* 2^n *empty membranes labelled by 2.*

Proof. (a) is going to be demonstrated by induction on l

- The base case $l = 0$ is going to be demonstrated by induction on k
  (a_0) The base case $k = 1$ is trivial because:
  - at configuration $\mathcal{C}_{2n}$ we have: $\mathcal{C}_{2n}(0) = \{\alpha_{2n}, \beta_{2n}\}$ and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{2n}(\varphi)$, an object γ_{2n} and p copies of T_i and F_i, being $1 \leq i \leq n$; and 2^n membranes labelled by 2 containing a different subset of objects $r_{i,2n-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, the corresponding truth assignment of the branch. Then, configuration $\mathcal{C}_{2n}$ yields configuration $\mathcal{C}_{2n+1}$ by applying the rules:
    $[\ t_{i,2n}\ ]_2 \to t_{i,2n+1}[\ \ ]_2$
    $[\ f_{i,2n}\ ]_2 \to f_{i,2n+1}[\ \ ]_2$

$$\left.\begin{array}{l}[\ t_{i,2n+1-i} \to t_{i,2n+2-i}\ ]_2\\ [\ f_{i,2n+1-i} \to f_{i,2n+2-1}\ ]_2\end{array}\right\} \text{ for } 2 \le i \le n$$

$$[\ \alpha_{2n} \to \alpha_{2n+1}\ ]_0$$

$$[\ \beta_{2n} \to \beta_{2n+1}\ ]_0$$

$$[\ \gamma_{2n} \to \gamma_{2n+1}\ ]_1$$

$$\left.\begin{array}{l}[\ x_{i,j,2n} \to x_{i,j,2n+1}\ ]_1\\ [\ \overline{x}_{i,j,2n} \to \overline{x}_{i,j,2n+1}\ ]_1\\ [\ x^*_{i,j,2n} \to x^*_{i,j,2n+1}\ ]_1\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p$$

Thus, $\mathcal{C}_{2n+1}(0) = \{\alpha_{2n+1}, \beta_{2n+1}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{2n+1}(\varphi)$, an object γ_{2n+1}, p copies of T_i and F_i, being $1 \le i \le n$ and an object $r_{1,2n+1}$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing a different subset of objects $r_{i,2n-i+2}$, $2 \le i \le n$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k $(1 \le k \le n)$
  - $\mathcal{C}_{2n+k}(0) = \{\alpha_{2n+k}, \beta_{2n+k}\}$
  - In $\mathcal{C}_{2n+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{2n+k}(\varphi)$;
    - ⋆ an object γ_{2n+k};
    - ⋆ p copies of every T_i and F_i, $1 \le i \le n$; and
    - ⋆ objects $r_{i,2n+k-i+1}$, $1 \le i \le k$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+k}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,2n+k-i+1}$, $k+1 \le i \le n$, being $r \in \{t, f\}$.

  Then, configuration $\mathcal{C}_{2n+k}$ yields configuration $\mathcal{C}_{2n+k+1}$ by applying the rules:

$$[\ t_{k+1,2n}\ ]_2 \to t_{k+1,2n+1}[\ \ ]_2$$

$$[\ f_{k+1,2n}\ ]_2 \to f_{k+1,2n+1}[\ \ ]_2$$

$$\left.\begin{array}{l}[\ t_{i,2n+k-i+1} \to t_{i,2n+k-i+2}\ ]_2\\ [\ f_{i,2n+k-i+1} \to f_{i,2n+k-i+2}\ ]_2\end{array}\right\} \text{ for } k+2 \le i \le n$$

$$\left.\begin{array}{l}[\ t_{i,2n+k-i+1} \to t_{i,2n+k-i+2}\ ]_1\\ [\ f_{i,2n+k-i+1} \to f_{i,2n+k-i+2}\ ]_1\end{array}\right\} \text{ for } 1 \le i \le k$$

$$[\ \alpha_{2n+k} \to \alpha_{2n+k+1}\ ]_0$$

$$[\ \beta_{2n+k} \to \beta_{2n+k+1}\ ]_0$$

$$[\ \gamma_{2n+k} \to \gamma_{2n+k+1}\ ]_1$$

$$\left.\begin{array}{l}[\ x_{i,j,2n+k} \to x_{i,j,2n+k+1}\ ]_1\\ [\ \overline{x}_{i,j,2n+k} \to \overline{x}_{i,j,2n+k+1}\ ]_1\\ [\ x^*_{i,j,2n+k} \to x^*_{i,j,2n+k+1}\ ]_1\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p$$

Therefore, the following holds

- $\mathcal{C}_{2n+k+1}(0) = \{\alpha_{2n+k+1}, \beta_{2n+k+1}\}$
- In $\mathcal{C}_{2n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the input multiset $cod_{2n+k+1}(\varphi)$;
  - ⋆ an object γ_{2n+k+1};

  - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$; and
  - ⋆ objects $r_{i,2n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.
- In $\mathcal{C}_{2n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,2n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.

(a_1) The base case $k = 1$ is trivial because:

- at configuration $\mathcal{C}_{3n}$ we have $\mathcal{C}_{3n}(0) = \{\alpha_{3n}, \beta_{3n}\}$ and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{3n}(\varphi)$, an object γ_{3n}, p copies of T_i and F_i, being $1 \leq i \leq n$ and a different subset of objects $r_{i,3n+1-i}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the corresponding truth assignment of the branch; and 2^n empty membranes labelled by 2. Then, configuration $\mathcal{C}_{3n}$ yields configuration $\mathcal{C}_{3n+1}$ by applying the rules:

$$
\begin{array}{l}
t_{1,3n}\ F_1[\ \ ]_2 \to [\ t_{1,3n+1}\ ]_2 \\
f_{1,3n}\ T_1[\ \ ]_2 \to [\ f_{1,3n+1}\ ]_2 \\
\left.\begin{array}{l} [\ t_{i,3n-i+1} \to t_{i,3n-i+2}\ ]_1 \\ [\ f_{i,3n-i+1} \to f_{i,3n-i+2}\ ]_1 \end{array}\right\} \text{ for } 2 \leq i \leq n \\
[\ \alpha_{3n} \to \alpha_{3n+1}\ ]_0 \\
[\ \beta_{3n} \to \beta_{3n+1}\ ]_0 \\
[\ \gamma_{3n} \to \gamma_{3n+1}\ ]_1 \\
\left.\begin{array}{l} [\ x_{i,j,3n} \to x_{i,j,3n+1}\ ]_1 \\ [\ \overline{x}_{i,j,3n} \to \overline{x}_{i,j,3n+1}\ ]_1 \\ [\ x^*_{i,j,3n} \to x^*_{i,j,3n+1}\ ]_1 \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

  Thus, $\mathcal{C}_{3n+1}(0) = \{\alpha_{3n+1}, \beta_{3n+1}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{3n+1}(\varphi)$, an object γ_{3n+1}, p copies of T_i and F_i, being $2 \leq i \leq n$, and $p-1$ copies of T_1 (resp. F_1) if we have its corresponding f_1 (resp. t_1) object in that branch, p copies otherwise, and a different subset of objects $r_{i,3n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing an object $r_{1,3n+1}$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k ($1 \leq k \leq n$)
  - $\mathcal{C}_{3n+k}(0) = \{\alpha_{3n+k}, \beta_{3n+k}\}$
  - In $\mathcal{C}_{3n+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{3n+k}(\varphi)$;
    - ⋆ an object γ_{3n+k};
    - ⋆ p copies of every T_i and F_i, if $k+1 \leq i \leq n$ or their corresponding t_i or f_i is assigned to that branch, $p-1$ copies otherwise; and
    - ⋆ objects $r_{i,3n+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{3n+k}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,3n+k-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{3n+k}$ yields configuration $\mathcal{C}_{3n+k+1}$ by applying the rules:

$t_{k+1,3n} F_k[\ \ ]_2 \to [\ t_{k+1,3n+1}\ ]_2$
$f_{k+1,3n} T_k[\ \ ]_2 \to [\ f_{k+1,3n+1}\ ]_2$
$\left.\begin{array}{l}[\ t_{i,3n+k-i+1} \to t_{i,3n+k-i+2}\ ]_1 \\ [\ f_{i,3n+k-i+1} \to f_{i,3n+k-i+2}\ ]_1\end{array}\right\}$ for $k+2 \leq i \leq n$
$\left.\begin{array}{l}[\ t_{i,3n+k-i+1} \to t_{i,3n+k-i+2}\ ]_2 \\ [\ f_{i,3n+k-i+1} \to f_{i,3n+k-i+2}\ ]_2\end{array}\right\}$ for $1 \leq i \leq k$
$[\ \alpha_{3n+k} \to \alpha_{3n+k+1}\ ]_0$
$[\ \beta_{3n+k} \to \beta_{3n+k+1}\ ]_0$
$[\ \gamma_{3n+k} \to \gamma_{3n+k+1}\ ]_1$
$\left.\begin{array}{l}[\ x_{i,j,3n+k} \to x_{i,j,3n+k+1}\ ]_1 \\ [\ \overline{x}_{i,j,3n+k} \to \overline{x}_{i,j,3n+k+1}\ ]_1 \\ [\ x^*_{i,j,3n+k} \to x^*_{i,j,3n+k+1}\ ]_1\end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

Therefore, the following holds

- $\mathcal{C}_{3n+k+1}(0) = \{\alpha_{3n+k+1}, \beta_{3n+k+1}\}$
- In $\mathcal{C}_{3n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the input multiset $cod_{3n+k+1}(\varphi)$;
  - ⋆ an object γ_{3n+k+1};
  - ⋆ p copies of every T_i and F_i, if $k+2 \leq i \leq n$ or the corresponding t_i or f_i is assigned to that branch, $p-1$ copies otherwise; and
  - ⋆ objects $r_{i,3n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.
- In $\mathcal{C}_{3n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,3n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for l $(0 \leq l \leq p-1)$

  (a_0) The base case $k = 1$ is trivial because:
  - at configuration $\mathcal{C}_{2n+(l+1)n}$[1] we have: $\mathcal{C}_{2n+(l+1)n}(0) = \{\alpha_{2n+(l+1)n}, \beta_{2n+(l+1)n}\}$ and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{2n+(l+1)n}(\varphi)$, an object $\gamma_{2n+(l+1)n}$ and p copies of T_i and F_i, being $1 \leq i \leq n$, and $p-l$ copies for T_i (resp. F_i) objects that are in a branch with an object f_i (resp. t_i); and 2^n membranes labelled by 2 containing a different subset of objects $r_{i,2n+(l+1)n-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, the corresponding truth assignment of the branch. Then, configuration $\mathcal{C}_{2n+(l+1)n}$ yields configuration $\mathcal{C}_{2n+(l+1)n+1}$ by applying the rules:
    $[\ t_{i,2n+(l+1)n}\ ]_2 \to t_{i,2n+(l+1)n+1}[\ \ ]_2$
    $[\ f_{i,2n+(l+1)n}\ ]_2 \to f_{i,2n+(l+1)n+1}[\ \ ]_2$
    $\left.\begin{array}{l}[\ t_{i,2n+(l+1)n+1-i} \to t_{i,2n+(l+1)n+2-i}\ ]_2 \\ [\ f_{i,2n+(l+1)n+1-i} \to f_{i,2n+(l+1)n+2-i}\ ]_2\end{array}\right\}$ for $2 \leq i \leq n$

[1] Note that $(l+1)n = ln + n$, and it has been demonstrated in the first step of the induction that it is correct.

$$\left.\begin{array}{l}
[\ \alpha_{2n+(l+1)n} \rightarrow \alpha_{2n+(l+1)n+1}\ ]_0 \\
[\ \beta_{2n+(l+1)n} \rightarrow \beta_{2n+(l+1)n+1}\ ]_0 \\
[\ \gamma_{2n+(l+1)n} \rightarrow \gamma_{2n+(l+1)n+1}\ ]_1 \\
[\ x_{i,j,2n+(l+1)n} \rightarrow x_{i,j,2n+(l+1)n+1}\ ]_1 \\
[\ \overline{x}_{i,j,2n+(l+1)n} \rightarrow \overline{x}_{i,j,2n+(l+1)n+1}\ ]_1 \\
[\ x^*_{i,j,2n+(l+1)n} \rightarrow x^*_{i,j,2n+(l+1)n+1}\ ]_1
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p$$

Thus, $\mathcal{C}_{2n+(l+1)n+1}(0) = \{\alpha_{2n+(l+1)n+1}, \beta_{2n+(l+1)n+1}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{2n+(l+1)n+1}(\varphi)$, an object $\gamma_{2n+(l+1)n+1}$, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p - l$ copies of F_i (resp. T_i) and an object $r_{1,2n+(l+1)n+1}$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing a different subset of objects $r_{i,2n+(l+1)n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k $(1 \leq k \leq n)$
  - $\mathcal{C}_{2n+(l+1)n+k}(0) = \{\alpha_{2n+(l+1)n+k}, \beta_{2n+(l+1)n+k}\}$
  - In $\mathcal{C}_{2n+(l+1)n+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{2n+(l+1)n+k}(\varphi)$;
    - ⋆ an object $\gamma_{2n+(l+1)n+k}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p - l$ copies of F_i (resp. T_i); and
    - ⋆ objects $r_{i,2n+(l+1)n+k-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+(l+1)n+k}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,2n+(l+1)n+k-i+1}$, $k + 1 \leq i \leq n$, being $r \in \{t, f\}$.

  Then, configuration $\mathcal{C}_{2n+k}$ yields configuration $\mathcal{C}_{2n+(l+1)n+k+1}$ by applying the rules:

$$\left.\begin{array}{l}
[\ t_{k+1,2n+(l+1)n}\ ]_2 \rightarrow t_{k+1,2n+(l+1)n+1}[\ \ ]_2 \\
[\ f_{k+1,2n+(l+1)n}\ ]_2 \rightarrow f_{k+1,2n+(l+1)n+1}[\ \ ]_2 \\
[\ t_{i,2n+(l+1)n+k-i+1} \rightarrow t_{i,2n+k-i+2}\ ]_2 \\
[\ f_{i,2n+(l+1)n+k-i+1} \rightarrow f_{i,2n+k-i+2}\ ]_2
\end{array}\right\} \text{for } k+2 \leq i \leq n$$

$$\left.\begin{array}{l}
[\ t_{i,2n+(l+1)n+k-i+1} \rightarrow t_{i,2n+(l+1)n+k-i+2}\ ]_1 \\
[\ f_{i,2n+(l+1)n+k-i+1} \rightarrow f_{i,2n+(l+1)n+k-i+2}\ ]_1
\end{array}\right\} \text{for } 1 \leq i \leq k$$

$$\left.\begin{array}{l}
[\ \alpha_{2n+(l+1)n+k} \rightarrow \alpha_{2n+(l+1)n+k+1}\ ]_0 \\
[\ \beta_{2n+(l+1)n+k} \rightarrow \beta_{2n+(l+1)n+k+1}\ ]_0 \\
[\ \gamma_{2n+(l+1)n+k} \rightarrow \gamma_{2n+(l+1)n+k+1}\ ]_1 \\
[\ x_{i,j,2n+(l+1)n+k} \rightarrow x_{i,j,2n+(l+1)n+k+1}\ ]_1 \\
[\ \overline{x}_{i,j,2n+(l+1)n+k} \rightarrow \overline{x}_{i,j,2n+(l+1)n+k+1}\ ]_1 \\
[\ x^*_{i,j,2n+(l+1)n+k} \rightarrow x^*_{i,j,2n+(l+1)n+k+1}\ ]_1
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p$$

  Therefore, the following holds
  - $\mathcal{C}_{2n+(l+1)n+k+1}(0) = \{\alpha_{2n+(l+1)n+k+1}, \beta_{2n+(l+1)n+k+1}\}$

- In $\mathcal{C}_{2n+(l+1)n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the input multiset $cod_{2n+(l+1)n+k+1}(\varphi)$;
  - ⋆ an object $\gamma_{2n+(l+1)n+k+1}$;
  - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i); and
  - ⋆ objects $r_{i,2n+(l+1)n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.
- In $\mathcal{C}_{2n+(l+1)n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,2n+(l+1)n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.

(a_1) The base case $k = 1$ is trivial because:

- at configuration $\mathcal{C}_{3n+(l+1)n}$ we have $\mathcal{C}_{3n+(l+1)n}(0) = \{\alpha_{3n+(l+1)n}, \beta_{3n+(l+1)n}\}$ and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{3n+(l+1)n}(\varphi)$, an object $\gamma_{3n+(l+1)n}$, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i) and a different subset of objects $r_{i,3n+(l+1)n-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the corresponding truth assignment of the branch; and 2^n empty membranes labelled by 2. Then, configuration $\mathcal{C}_{3n+(l+1)n}$ yields configuration $\mathcal{C}_{3n+(l+1)n+1}$ by applying the rules:

$$
\begin{array}{l}
t_{1,3n+(l+1)n}\ F_1[\ \ ]_2 \to [\ t_{1,3n+(l+1)n+1}\ ]_2 \\
f_{1,3n+(l+1)n}\ T_1[\ \ ]_2 \to [\ f_{1,3n+(l+1)n+1}\ ]_2 \\
\left.\begin{array}{l}
[\ t_{i,3n+(l+1)n-i+1} \to t_{i,3n+(l+1)n-i+2}\ ]_1 \\
[\ f_{i,3n+(l+1)n-i+1} \to f_{i,3n+(l+1)n-i+2}\ ]_1
\end{array}\right\} \text{ for } 2 \leq i \leq n \\
[\ \alpha_{3n+(l+1)n} \to \alpha_{3n+(l+1)n+1}\ ]_0 \\
[\ \beta_{3n+(l+1)n} \to \beta_{3n+(l+1)n+1}\ ]_0 \\
[\ \gamma_{3n+(l+1)n} \to \gamma_{3n+(l+1)n+1}\ ]_1 \\
\left.\begin{array}{l}
[\ x_{i,j,3n+(l+1)n} \to x_{i,j,3n+(l+1)n+1}\ ]_1 \\
[\ \overline{x}_{i,j,3n+(l+1)n} \to \overline{x}_{i,j,3n+(l+1)n+1}\ ]_1 \\
[\ x^*_{i,j,3n+(l+1)n} \to x^*_{i,j,3n+(l+1)n+1}\ ]_1
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

  Thus, $\mathcal{C}_{3n+(l+1)n+1}(0) = \{\alpha_{3n+(l+1)n+1}, \beta_{3n+(l+1)n+1}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{3n+(l+1)n+1}(\varphi)$, an object $\gamma_{3n+(l+1)n+1}$, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i) if $k+1 \leq i \leq n$, $p-l-1$ otherwise, and a different subset of objects $r_{i,3n+(l+1)n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing an object $r_{1,3n+(l+1)n+1}$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k ($1 \leq k \leq n$)
  - $\mathcal{C}_{3n+(l+1)n+k}(0) = \{\alpha_{3n+(l+1)n+k}, \beta_{3n+(l+1)n+k}\}$
  - In $\mathcal{C}_{3n+(l+1)n+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{3n+(l+1)n+k}(\varphi)$;

  - ⋆ an object $\gamma_{3n+(l+1)n+k}$;
  - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i) if $k+1 \leq i \leq n$, $p-l-1$ otherwise; and
  - ⋆ objects $r_{i,3n+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.
- In $\mathcal{C}_{3n+(l+1)n+k}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,3n+(l+1)n-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.

  Then, configuration $\mathcal{C}_{3n+(l+1)n+k}$ yields configuration $\mathcal{C}_{3n+(l+1)n+k+1}$ by applying the rules:

$$
\begin{array}{l}
t_{k+1,3n+(l+1)n}\ F_k[\ \ ]_2 \to [\ t_{k+1,3n+(l+1)n+1}\ ]_2 \\
f_{k+1,3n+(l+1)n}\ T_k[\ \ ]_2 \to [\ f_{k+1,3n+(l+1)n+1}\ ]_2 \\
\left.\begin{array}{l}
[\ t_{i,3n+(l+1)n+k-i+1} \to t_{i,3n+(l+1)n+k-i+2}\ ]_1 \\
[\ f_{i,3n+(l+1)n+k-i+1} \to f_{i,3n+(l+1)n+k-i+2}\ ]_1
\end{array}\right\} \text{for } k+2 \leq i \leq n \\
\left.\begin{array}{l}
[\ t_{i,3n+(l+1)n+k-i+1} \to t_{i,3n+(l+1)n+k-i+2}\ ]_2 \\
[\ f_{i,3n+(l+1)n+k-i+1} \to f_{i,3n+(l+1)n+k-i+2}\ ]_2
\end{array}\right\} \text{for } 1 \leq i \leq k \\
[\ \alpha_{3n+(l+1)n+k} \to \alpha_{3n+(l+1)n+k+1}\ ]_0 \\
[\ \beta_{3n+(l+1)n+k} \to \beta_{3n+(l+1)n+k+1}\ ]_0 \\
[\ \gamma_{3n+(l+1)n+k} \to \gamma_{3n+(l+1)n+k+1}\ ]_1 \\
\left.\begin{array}{l}
[\ x_{i,j,3n+(l+1)n+k} \to x_{i,j,3n+(l+1)n+k+1}\ ]_1 \\
[\ \overline{x}_{i,j,3n+(l+1)n+k} \to \overline{x}_{i,j,3n+(l+1)n+k+1}\ ]_1 \\
[\ x^*_{i,j,3n+(l+1)n+k} \to x^*_{i,j,3n+(l+1)n+k+1}\ ]_1
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

  Therefore, the following holds
  - $\mathcal{C}_{3n+(l+1)n+k+1}(0) = \{\alpha_{3n+(l+1)n+k+1}, \beta_{3n+(l+1)n+k+1}\}$
  - In $\mathcal{C}_{3n+(l+1)n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{3n+(l+1)n+k+1}(\varphi)$;
    - ⋆ an object $\gamma_{3n+(l+1)n+k+1}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i) if $k+2 \leq i \leq n$, $p-l-1$ otherwise; and
    - ⋆ objects $r_{i,3n+(l+1)n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{3n+(l+1)n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains a different subset of objects $r_{i,3n+(l+1)n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.
- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{n+2np-1}$[2] holds:
  - $\mathcal{C}_{n+2np-1}(0) = \{\alpha_{n+2np-1}, \beta_{n+2np-1}\}$
  - In $\mathcal{C}_{n+2np-1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{n+2np-1}(\varphi)$;
    - ⋆ an object $\gamma_{n+2np-1}$;

[2] Note that $n + 2np - 1 = 2n + 2n(p-1) + (n-1)$

  - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy otherwise; and
  - ⋆ objects $r_{i,n+2np-i}$, $1 \leq i \leq n-1$
- In $\mathcal{C}_{n+2np-1}$ there are 2^n membranes labelled by 2 such that each of them contains an object $r_{n,2np}$, being $r \in \{t, f\}$.

  Then, configuration $\mathcal{C}_{n+2np-1}$ yields $\mathcal{C}_{n+2np}$ by applying the rules:

$$\left.\begin{array}{l} [\ t_{n,2np}\ ]_2 \rightarrow t_{n,2np+1}[\ \ ]_2 \\ [\ f_{n,2np}\ ]_2 \rightarrow f_{n,2np+1}[\ \ ]_2 \\ [\ t_{i,n+2np-i} \rightarrow t_{i,n+2np-i+1}\ ]_1 \\ [\ f_{i,n+2np-i} \rightarrow f_{i,n+2np-i}\ ]_1 \end{array}\right\} \text{ for } 1 \leq i \leq n-1$$

$$[\ \alpha_{n+2np-1} \rightarrow \alpha_{n+2np}\ ]_0$$
$$[\ \beta_{n+2np-1} \rightarrow \beta_{n+2np}\ ]_0$$
$$[\ \gamma_{n+2np-1} \rightarrow \gamma_{n+2np}\ ]_1$$

$$\left.\begin{array}{l} [\ x_{i,j,n+2np-1} \rightarrow x_{i,j,n+2np}\ ]_1 \\ [\ \overline{x}_{i,j,n+2np-1} \rightarrow \overline{x}_{i,j,n+2np}\ ]_1 \\ [\ x^*_{i,j,n+2np-1} \rightarrow x^*_{i,j,n+2np}\ ]_1 \end{array}\right\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p$$

  Then, we have $\mathcal{C}_{n+2np}(0) = \{\alpha_{n+2np}, \beta_{n+2np}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{n+2np}(\varphi)$, an object γ_{n+2np}, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy otherwise and a different multiset of objects $r_{i,n+2np-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the truth assignment associated with the branch; and 2^n empty membranes labelled by 2.

□

Proposition 3. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a) For each k ($1 \leq k \leq n-1$) at configuration $\mathcal{C}_{n+2np+k}$ we have the following:
- *$\mathcal{C}_{n+2np+k}(0) = \{\alpha_{n+2np+k}, \beta_{n+2np+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains*
  - ⋆ *the input multiset $cod_{n+2np+k}(\varphi)$;*
  - ⋆ *an object $\gamma_{n+2np+k}$;*
  - ⋆ *p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+1 \leq i \leq n$; and*
  - ⋆ *objects $r_{i,n+2np+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.*
- *there are 2^n membranes labelled by 2 such that each of them contains k objects #*

(b) $\mathcal{C}_{2n+2np}(0) = \{\alpha_{2n+2np}, \beta_{2n+2np}\}$, and in $\mathcal{C}_{2n+2np}$ there are 2^n membranes labelled by 1, such that each of them contains the input multiset $cod_{2n+2np}(\varphi)$, an object γ_{2n+2np}, p copies of every T_i and F_i, $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding t_i or f_i object; and 2^n membranes labelled by 2, such that each of them contains n objects #.

Proof. (a) is going to be demonstrated by induction on k

- the base case $k = 1$ is trivial because:
  - at $\mathcal{C}_{n+2np}$ we have $\mathcal{C}_{n+2np}(0) = \{\alpha_{n+2np}, \beta_{n+2np}\}$ and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{n+2np}(\varphi)$, an object γ_{n+2np} p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy otherwise and a different multiset of objects $r_{i,n+2np-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the truth assignment associated with the branch; and 2^n empty membranes labelled by 2. Then, configuration $\mathcal{C}_{n+2np}$ yields $\mathcal{C}_{n+2np+1}$ by applying the rules.

$$
\begin{array}{l}
t_{1,n+2np}\ F_1[\ \ ]_2 \to [\ \#\ ]_2 \\
f_{1,n+2np}\ T_1[\ \ ]_2 \to [\ \#\ ]_2 \\
\left.\begin{array}{l}
[\ t_{i,n+2np-i+1} \to t_{i,n+2np-i+2}\ ]_1 \\
[\ f_{i,n+2np-i+1} \to f_{i,n+2np-i+2}\ ]_1
\end{array}\right\} \text{ for } 2 \leq i \leq n \\
[\ \alpha_{n+2np} \to \alpha_{n+2np+1}\ ]_0 \\
[\ \beta_{n+2np} \to \beta_{n+2np+1}\ ]_0 \\
[\ \gamma_{n+2np} \to \gamma_{n+2np+1}\ ]_1 \\
\left.\begin{array}{l}
[\ x_{i,j,n+2np} \to x_{i,j,n+2np+1}\ ]_1 \\
[\ \overline{x}_{i,j,n+2np} \to \overline{x}_{i,j,n+2np+1}\ ]_1 \\
[\ x^*_{i,j,n+2np} \to x^*_{i,j,n+2np+1}\ ]_1
\end{array}\right\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

    Thus, $\mathcal{C}_{n+2np+1}(0) = \{\alpha_{n+2np+1}, \beta_{n+2np+1}\}$, and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{n+2np+1}(\varphi)$, an object $\gamma_{n+2np+1}$, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+2 \leq i \leq n$ and objects $r_{i,n+2np-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing an object $\#$.
- Supposing, by induction, result is true for k $(1 \leq k \leq n-1)$
  - $\mathcal{C}_{n+2np+k}(0) = \{\alpha_{n+2np+k}, \beta_{n+2np+k}\}$
  - In $\mathcal{C}_{n+2np+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{n+2np+k}(\varphi)$;
    - ⋆ an object $\gamma_{n+2np+k}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+1 \leq i \leq n$; and
    - ⋆ objects $r_{i,n+2np+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{n+2np+k}$ there are 2^n membranes labelled by 2 such that each of them contains k objects $\#$.

    Then, configuration $\mathcal{C}_{n+2np+k}$ yields configuration $\mathcal{C}_{n+2np+k+1}$ by applying the rules:

$$
\begin{array}{l}
t_{k+1,n+2np}\ F_1[\ \ ]_2 \to [\ \#\ ]_2 \\
f_{k+1,n+2np}\ T_1[\ \ ]_2 \to [\ \#\ ]_2 \\
\left.\begin{array}{l}
[\ t_{i,n+2np+k-i+1} \to t_{i,n+2np+k-i+2}\ ]_1 \\
[\ f_{i,n+2np+k-i+1} \to f_{i,n+2np+k-i+2}\ ]_1
\end{array}\right\} \text{ for } 2 \leq i \leq n
\end{array}
$$

$[\ \alpha_{n+2np+k} \to \alpha_{n+2np+k+1}\ ]_0$
$[\ \beta_{n+2np+k} \to \beta_{n+2np+k+1}\ ]_0$
$[\ \gamma_{n+2np+k} \to \gamma_{n+2np+k+1}\ ]_1$
$\left.\begin{array}{l}[\ x_{i,j,n+2np+k} \to x_{i,j,n+2np+k+1}\ ]_1\\ [\ \overline{x}_{i,j,n+2np+k} \to \overline{x}_{i,j,n+2np+k+1}\ ]_1\\ [\ x^*_{i,j,n+2np+k} \to x^*_{i,j,n+2np+k+1}\ ]_1\end{array}\right\}$ for $1 \le i \le n, 1 \le j \le p$

Therefore, the following holds

- $\mathcal{C}_{n+2np+k+1}(0) = \{\alpha_{n+2np+k+1}, \beta_{n+2np+k+1}\}$
- In $\mathcal{C}_{n+2np+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the input multiset $cod_{n+2np+k+1}(\varphi)$;
  - ⋆ an object $\gamma_{n+2np+k+1}$;
  - ⋆ p copies of T_i (resp. F_i) being $1 \le i \le n$ if their corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+2 \le i \le n$; and
  - ⋆ objects $r_{i,n+2np+k-i+2}$, $k+2 \le i \le n$, being $r \in \{t, f\}$.
- In $\mathcal{C}_{n+2np+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains $k+1$ objects #.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{2n+2np-1}$[3] holds:
  - $\mathcal{C}_{2n+2np-1}(0) = \{\alpha_{2n+2np-1}, \beta_{2n+2np-1}\}$
  - In $\mathcal{C}_{2n+2np-1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the input multiset $cod_{2n+2np-1}(\varphi)$;
    - ⋆ an object $\gamma_{2n+2np-1}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \le i \le n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_n (resp. T_n); and
    - ⋆ an object $r_{n,n+2np}$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+2np-1}$ there are 2^n membranes labelled by 2 such that each of them contains $n-1$ objects #.

    Then, configuration $\mathcal{C}_{2n+2np-1}$ yields configuration $\mathcal{C}_{2n+2np}$ by applying the rules:

    $t_{n,n+2np}\ F_1[\ \ ]_2 \to [\ \#\ ]_2$
    $f_{n,n+2np}\ T_1[\ \ ]_2 \to [\ \#\ ]_2$
    $[\ \alpha_{2n+2np-1} \to \alpha_{2n+2np}\ ]_0$
    $[\ \beta_{2n+2np-1} \to \beta_{2n+2np}\ ]_0$
    $[\ \gamma_{2n+2np-1} \to \gamma_{2n+2np}\ ]_1$
    $\left.\begin{array}{l}[\ x_{i,j,2n+2np-1} \to x_{i,j,2n+2np}\ ]_1\\ [\ \overline{x}_{i,j,2n+2np-1} \to \overline{x}_{i,j,2n+2np}\ ]_1\\ [\ x^*_{i,j,2n+2np-1} \to x^*_{i,j,2n+2np}\ ]_1\end{array}\right\}$ for $1 \le i \le n, 1 \le j \le p$

    Therefore, the following holds
  - $\mathcal{C}_{2n+2np}(0) = \{\alpha_{2n+2np}, \beta_{2n+2np}\}$

[3] Note that $2n + 2np - 1 = n + 2np + (n-1)$

- In $\mathcal{C}_{2n+2np}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the input multiset $cod_{2n+2np}(\varphi)$;
  - ⋆ an object γ_{2n+2np}; and
  - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch.
- In $\mathcal{C}_{2n+2np}$ there are 2^n membranes labelled by 2 such that each of them contains n objects #.

□

5.2 First checking stage

At this stage, we try to determine the clauses satisfied for the truth assignment encoded by each branch. For that, rules from **5.5** will be applied in such manner that in the m-th step, being $m = ln+k$ ($1 \leq k \leq n$, $0 \leq l \leq p-1$), clause C_{l+1} will be evaluated with the k-th variable of the formula. This stage will take exactly np steps.

Proposition 4. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a) For each k ($1 \leq k \leq n$) and l ($0 \leq l \leq p-1$) at configuration $\mathcal{C}_{2n+2np+ln+k}$ we have the following:
- *$\mathcal{C}_{2n+2np+ln+k}(0) = \{\alpha_{2n+2np+ln+k}, \beta_{2n+2np+ln+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains*
  - ⋆ *the $(n-k)$-th last elements of $cod_{2n+2np+ln+k}(\varphi)^{l+1}_{l+1}$;*
  - ⋆ *the input multiset $cod_{2n+2np+ln+k}(\varphi)^{p}_{l+2}$;*
  - ⋆ *an object $\gamma_{2n+2np+ln+k}$; and*
  - ⋆ *$p-l$ copies of objects T_i or F_i, $k+1 \leq i \leq n$, $p-l-1$ copies otherwise, corresponding to the truth assignment assigned to the branch.*
- *There are 2^n membranes labelled by 2 such that each of them contains*
  - ⋆ *m objects $c_{j,t}$ ($1 \leq j \leq l+1$, $0 \leq t \leq ln+k-1$), that is, clauses that have been satisfied by any variable; and*
  - ⋆ *$n+ln+k-m$ objects #.*

(b) $\mathcal{C}_{2n+3np}(0) = \{\alpha_{2n+3np}, \beta_{2n+3np}\}$, and in $\mathcal{C}_{2n+3np}$ there are 2^n membranes labelled by 1, such that each of them contains an object γ_{2n+3np}; and 2^n membranes labelled by 2 such that each of them contains m objects $c_{j,t}$ ($1 \leq j \leq p$, $0 \leq t \leq np-1$), that is, the clauses satisfied by any variable and $n+np-m$ objects #.

Proof. (*a*) is going to be demonstrated by induction on l

- The base case $l = 0$ is goig to be demonstrated by induction on k
  - The base case $k = 1$ is trivial because:

- at configuration $\mathcal{C}_{2n+2np}$ we have: $\mathcal{C}_{2n+2np}(0) = \{\alpha_{2n+2np}, \beta_{2n+2np}\}$ and there exist 2^n membranes labelled by 1, such that each of them contains the input multiset $cod_{2n+2np}(\varphi)$, an object γ_{2n+2np} and p copies of objects T_i and F_i, $1 \leq i \leq n$, representing the correspondent truth assignment to the branch; and 2^n membranes labelled by 2 such that each of them contains n objects #. Then, configuration $\mathcal{C}_{2n+2np}$ yields configuration $\mathcal{C}_{2n+2np+1}$ by applying the rules:

$$
\begin{array}{l}
T_1\ x_{1,1,2n+2np}[\ \ ]_2 \longrightarrow [c_{1,0}]_2 \\
T_1\ \overline{x}_{1,1,2n+2np}[\ \ ]_2 \longrightarrow\ [\#]_2 \\
T_1\ x^*_{1,1,2n+2np}[\ \ ]_2 \longrightarrow\ [\#]_2 \quad {}^{4} \\
F_1\ x_{1,1,2n+2np}[\ \ ]_2 \longrightarrow\ [\#]_2 \\
F_1\ \overline{x}_{1,1,2n+2np}[\ \ ]_2 \longrightarrow\ [c_{1,0}]_2 \\
F_1\ x^*_{1,1,2n+2np}[\ \ ]_2 \longrightarrow\ [\#]_2 \\
[\ \alpha_{2n+2np} \rightarrow \alpha_{2n+2np+1}\ ]_0 \\
[\ \beta_{2n+2np} \rightarrow \beta_{2n+2np+1}\ ]_0 \\
[\ \gamma_{2n+2np} \rightarrow \gamma_{2n+2np+1}\ ]_1 \\
\left.\begin{array}{l}
[\ x_{i,j,2n+2np} \rightarrow x_{i,j,2n+2np+1}\ ]_1 \\
[\ \overline{x}_{i,j,2n+2np} \rightarrow \overline{x}_{i,j,2n+2np+1}\ ]_1 \\
[\ x^*_{i,j,2n+2np} \rightarrow x^*_{i,j,2n+2np+1}\ ]_1
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

  Thus, $\mathcal{C}_{2n+2np+1}(0) = \{\alpha_{2n+2np+1}, \beta_{2n+2np+1}\}$, and there exist 2^n membranes labelled by 1 containing the last $n-1$ elements of $cod_{2n+2np+1}(\varphi)^1_1$, the input multiset $cod_{2n+2np+1}(\varphi)^p_2$, p copies of T_i or F_i, being $2 \leq i \leq n$, and $p-1$ copies of T_1 or F_1; and 2^n membranes labelled by 2 containing n objects # and an object $c_{1,0}$ if the corresponding truth assignment makes true clause 1 with variable 1, another object # otherwise.
- Supposing, by induction, result is true for k $(1 \leq k \leq n)$
  - $\mathcal{C}_{2n+2np+k}(0) = \{\alpha_{2n+2np+k}, \beta_{2n+2np+k}\}$
  - In $\mathcal{C}_{2n+2np+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the $(n-k)$-th last elements of $cod_{2n+2np+k}(\varphi)^1_1$;
    - ⋆ the input multiset $cod_{2n+2np+k}(\varphi)^p_2$;
    - ⋆ an object $\gamma_{2n+2np+k}$; and
    - ⋆ p copies of objects T_i or F_i, $k+1 \leq i \leq n$, $p-1$ copies if $1 \leq i \leq k$, corresponding to the truth assignment assigned to the branch.
  - In $\mathcal{C}_{2n+2np+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ m objects $c_{1,t}$ $(0 \leq t \leq k-1)$, that is, the number of variables with the corresponding truth assignment that makes true the input formula φ; and
    - ⋆ $n+k-m$ objects #.

    Then, configuration $\mathcal{C}_{2n+2np+k}$ yields configuration $\mathcal{C}_{2n+2np+k+1}$ by applying the rules:

[4] If $k = 1, l = 0$, then $i = 1, j = 1$, so $2np + 2n + n(j-1) + (i-1) = 2n + 2np$

$T_k\ x_{k+1,1,2n+2np+k}[\quad]_2 \longrightarrow [c_{1,0}]_2$
$T_k\ \overline{x}_{k+1,1,2n+2np+k}[\quad]_2 \longrightarrow [\#]_2$
$T_k\ x^*_{k+1,1,2n+2np+k}[\quad]_2 \longrightarrow [\#]_2$ [5]
$F_k\ x_{k+1,1,2n+2np+k}[\quad]_2 \longrightarrow [\#]_2$
$F_k\ \overline{x}_{k+1,1,2n+2np+k}[\quad]_2 \longrightarrow [c_{1,0}]_2$
$F_k\ x^*_{k+1,1,2n+2np+k}[\quad]_2 \longrightarrow [\#]_2$
$[\ \alpha_{2n+2np+k} \to \alpha_{2n+2np+k+1}\ ]_0$
$[\ \beta_{2n+2np+k} \to \beta_{2n+2np+k+1}\ ]_0$
$[\ \gamma_{2n+2np+k} \to \gamma_{2n+2np+k+1}\ ]_1$
$\left.\begin{array}{l}[\ x_{i,j,2n+2np+k} \to x_{i,j,2n+2np+k+1}\ ]_1\\ [\ \overline{x}_{i,j,2n+2np+k} \to \overline{x}_{i,j,2n+2np+k+1}\ ]_1\\ [\ x^*_{i,j,2n+2np+k} \to x^*_{i,j,2n+2np+k+1}\ ]_1\end{array}\right\}$ for $1 \le i \le n, 1 \le j \le p$
$[\ c_{1,t} \to c_{1,t+1}\ ]_2$ for $0 \le t \le k-1$

Therefore, the following holds

- $\mathcal{C}_{2n+2np+k+1} = \{\alpha_{2n+2np+k+1}, \beta_{2n+2np+k+1}\}$
- In $\mathcal{C}_{2n+2np+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the $(n-k+1)$-th last elements of $cod_{2n+2np+k+1}(\varphi)^1_1$;
  - ⋆ the input multiset $cod_{2n+2np+k+1}(\varphi)^p_2$,
  - ⋆ an object $\gamma_{2n+2np+k+1}$; and
  - ⋆ p copies of objects T_i or F_i, $k+2 \le i \le n$, $p-1$ copies if $1 \le i \le k+1$, corresponding to the truth assignment assigned to the branch.
- In $\mathcal{C}_{2n+2np+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ m objects $c_{1,t}$ $(0 \le t \le k)$, that is, the number of variables with the corresponding truth assignment that makes true the clause $\mathcal{C}_1$; and
  - ⋆ $n+k+1-m$ objects $\#$.

- Supposing, by induction, result is true for l $(0 \le l \le p-1)$
  - The base case $k = 1$ is trivial because:
    - at configuration $\mathcal{C}_{2n+2np+(l+1)n}$ we have: $\mathcal{C}_{2n+2np+(l+1)n}(0) = \{\alpha_{2n+2np+(l+1)n}, \beta_{2n+2np+(l+1)n}\}$ and there exist 2^n membranes labelled by 1 containing the input multiset $cod_{2n+2np+(l+1)n}(\varphi)^p_{l+1}$, an object $\gamma_{2n+2np+(l+1)n}$ and $p-l$ copies of objects T_i or F_i, $1 \le i \le n$; and 2^n membranes labelled by 2 containing m objects $c_{j,t}$ $(1 \le j \le l,\ 0 \le t \le ln-1)$, that is, the number of variables with the corresponding truth assignment that makes true the clauses from $\mathcal{C}_1$ to $\mathcal{C}_l$ and $n+(l+1)n-m$ objects $\#$. Then, configuration $\mathcal{C}_{2n+2np+(l+1)n}$ yields configuration $\mathcal{C}_{2n+2np+(l+1)n+1}$ by applying the rules:

[5] If $l = 0$, then $i = k+1, j = 1$, so $2np + 2n + n(j-1) + (i-1) = 2n + 2np + k$

$$
\begin{array}{l}
T_1\ x_{1,1,2n+2np+(l+1)n}[\ \ ]_2 \longrightarrow [c_{l+1,0}]_2\\
T_1\ \overline{x}_{1,1,2n+2np+(l+1)n}[\ \ ]_2 \longrightarrow\ [\#]_2\\
T_1\ x^*_{1,1,2n+2np+(l+1)n}[\ \ ]_2 \longrightarrow\ [\#]_2\\
F_1\ x_{1,1,2n+2np+(l+1)n}[\ \ ]_2 \longrightarrow\ [\#]_2\\
F_1\ \overline{x}_{1,1,2n+2np+(l+1)n}[\ \ ]_2 \longrightarrow\ [c_{l+1,0}]_2\\
F_1\ x^*_{1,1,2n+2np+(l+1)n}[\ \ ]_2 \longrightarrow\ [\#]_2\\
[\ \alpha_{2n+2np+(l+1)n} \to \alpha_{2n+2np+(l+1)n+1}\ ]_0\\
[\ \beta_{2n+2np+(l+1)n} \to \beta_{2n+2np+(l+1)n+1}\ ]_0\\
[\ \gamma_{2n+2np+(l+1)n} \to \gamma_{2n+2np+(l+1)n+1}\ ]_1\\
\left.\begin{array}{l}
[\ x_{i,j,2n+2np+(l+1)n} \to x_{i,j,2n+2np+(l+1)n+1}\ ]_1\\
[\ \overline{x}_{i,j,2n+2np+(l+1)n} \to \overline{x}_{i,j,2n+2np+(l+1)n+1}\ ]_1\\
[\ x^*_{i,j,2n+2np+(l+1)n} \to x^*_{i,j,2n+2np+(l+1)n+1}\ ]_1
\end{array}\right\} \text{for } \begin{array}{l} 1 \le i \le n\\ 1 \le j \le p \end{array}\\
[\ c_{j,t} \to c_{j,t+1}\ ]_2 \text{ for } 1 \le j \le l+1, 0 \le t \le ln-1
\end{array}
$$

Thus, $\mathcal{C}_{2n+2np+(l+1)n+1}(0) = \{\alpha_{2n+2np+(l+1)n+1}, \beta_{2n+2np+(l+1)n+1}\}$, and there exist 2^n membranes labelled by 1 containing the last $n-1$ elements of $cod_{2n+2np+(l+1)n+1}(\varphi)^{l+1}_{l+1}$, the input multiset $cod_{2n+2np+(l+1)n+1}(\varphi)^{p}_{l+2}$, $p-l$ copies of T_i or F_i, being $2 \le i \le n$, and $p-l-1$ copies of T_1 or F_1; and 2^n membranes labelled by 2 containing m objects $c_{j,t}$ ($1 \le j \le ln$, $0 \le t \le ln$), that is, the number of variables with the corresponding truth assignment that makes true the clauses from $\mathcal{C}_1$ to $\mathcal{C}_{l+1}$ and $n+(l+1)n+1-m$ objects $\#$.

- Supposing, by induction, result is true for k ($1 \le k \le n$)
  - $\mathcal{C}_{2n+2np+(l+1)n+k}(0) = \{\alpha_{2n+2np+(l+1)n+k}, \beta_{2n+2np+(l+1)n+k}\}$
  - In $\mathcal{C}_{2n+2np+(l+1)n+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the $(n-k)$-th last elements of $cod_{2n+2np+(l+1)n+k}(\varphi)^{l+1}_{l+1}$;
    - ⋆ the input multiset $cod_{2n+2np+(l+1)n+k}(\varphi)^{p}_{l+2}$,
    - ⋆ an object $\gamma_{2n+2np+(l+1)n+k}$; and
    - ⋆ $p-l$ copies of objects T_i or F_i, $k+1 \le i \le n$, $p-l-1$ copies if $1 \le i \le k$, corresponding to the truth assignment assigned to the branch.
  - In $\mathcal{C}_{2n+2np+(l+1)n+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ m objects $c_{j,t}$ ($1 \le j \le l+1$, $0 \le t \le ln+k-1$), that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_{l+1}$; and
    - ⋆ $n+(l+1)n+k+1-m$ objects $\#$.

    Then, configuration $\mathcal{C}_{2n+2np+(l+1)n+k}$ yields configuration $\mathcal{C}_{2n+2np+(l+1)n+k+1}$ by applying the rules:

$T_k\ x_{1,1,2n+2np+(l+1)n+k}[\ \ ]_2 \longrightarrow [c_{l+1,0}]_2$
$T_k\ \overline{x}_{1,1,2n+2np+(l+1)n+k}[\ \ ]_2 \longrightarrow [\#]_2$
$T_k\ x^*_{1,1,2n+2np+(l+1)n+k}[\ \ ]_2 \longrightarrow [\#]_2$
$F_k\ x_{1,1,2n+2np+(l+1)n+k}[\ \ ]_2 \longrightarrow [\#]_2$
$F_k\ \overline{x}_{1,1,2n+2np+(l+1)n+k}[\ \ ]_2 \longrightarrow [c_{l+1,0}]_2$
$F_k\ x^*_{1,1,2n+2np+(l+1)n+k}[\ \ ]_2 \longrightarrow [\#]_2$
$[\ \alpha_{2n+2np+(l+1)n+k} \to \alpha_{2n+2np+(l+1)n+k+1}\ ]_0$
$[\ \beta_{2n+2np+(l+1)n+k} \to \beta_{2n+2np+(l+1)n+k+1}\ ]_0$
$[\ \gamma_{2n+2np+(l+1)n+k} \to \gamma_{2n+2np+(l+1)n+k+1}\ ]_1$

$$\left.\begin{array}{l} [\ x_{i,j,2n+2np+(l+1)n+k} \to x_{i,j,2n+2np+(l+1)n+k+1}\ ]_1 \\ [\ \overline{x}_{i,j,2n+2np+(l+1)n+k} \to \overline{x}_{i,j,2n+2np+(l+1)n+k+1}\ ]_1 \\ [\ x^*_{i,j,2n+2np+(l+1)n+k} \to x^*_{i,j,2n+2np+(l+1)n+k+1}\ ]_1 \end{array}\right\} \text{for } \begin{array}{c} 1 \le i \le n \\ 1 \le j \le p \end{array}$$

$[\ c_{j,t} \to c_{j,t+1}\ ]_2$ for $1 \le j \le l+1, 0 \le t \le ln+k-1$

Therefore, the following holds

- $\mathcal{C}_{2n+2np+(l+1)n+k+1}(0) = \{\alpha_{2n+2np+(l+1)n+k+1}, \beta_{2n+2np+(l+1)n+k+1}\}$
- In $\mathcal{C}_{2n+2np+(l+1)n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ the $(n-(k+1))$-th last elements of $cod_{2n+2np+(l+1)n+k+1}(\varphi)^{l+1}_{l+1}$,
  - ⋆ the input multiset $cod_{2n+2np+(l+1)n+k+1}(\varphi)^{p}_{l+1}$,
  - ⋆ an object $\gamma_{2n+2np+(l+1)n+k+1}$;
  - ⋆ $p-l$ copies of objects T_i or F_i, $k+2 \le i \le n$, $p-l-1$ copies if $1 \le i \le k+1$, corresponding to the truth assignment assigned to the branch.
- In $\mathcal{C}_{2n+2np+(l+1)n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ m objects $c_{j,t}$ $(1 \le j \le l+1,\ 0 \le t \le ln+k)$, that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_{l+1}$; and
  - ⋆ $n+(l+1)n+k+1-m$ objects #.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{2n+3np-1}$ [6] holds:
  - $\mathcal{C}_{2n+3np-1}(0) = \{\alpha_{2n+3np-1}, \beta_{2n+3np-1}\}$
  - In $\mathcal{C}_{2n+3np-1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ the last element of $cod_{2n+3np-1}(\varphi)^{p}_{p}$;
    - ⋆ an object $\gamma_{2n+3np-1}$; and
    - ⋆ an object T_n or F_n corresponding to the truth assignment assigned to the branch.
  - In $\mathcal{C}_{2n+3np-1}$ there are 2^n membranes labelled by 2 such that each of them contains

[6] Note that $2n+3np-1 = 2n+3n(p-1)+(n-1)$

- ⋆ m objects $c_{j,t}$ $(1 \leq j \leq p, 0 \leq t \leq np-2)$, that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_p$; and
- ⋆ $n + np - 1 - m$ objects $\#$.

Then, configuration $\mathcal{C}_{2n+3np-1}$ yields $\mathcal{C}_{2n+3np}$ by applying the rules:

$T_n\ x_{n,p,2n+3np-1}[\ \ ]_2 \longrightarrow [c_{p,0}]_2$
$T_n\ \overline{x}_{n,p,2n+3np-1}[\ \ ]_2 \longrightarrow [\#]_2$
$T_n\ x^*_{n,p,2n+3np-1}[\ \ ]_2 \longrightarrow [\#]_2$
$F_n\ x_{n,p,2n+3np-1}[\ \ ]_2 \longrightarrow [\#]_2$
$F_n\ \overline{x}_{n,p,2n+3np-1}[\ \ ]_2 \longrightarrow [c_{p,0}]_2$
$F_n\ x^*_{n,p,2n+3np-1}[\ \ ]_2 \longrightarrow [\#]_2$
$[\ \alpha_{2n+2np+(l+1)n+k} \rightarrow \alpha_{2n+2np+(l+1)n+k+1}\ ]_0$
$[\ \beta_{2n+2np+(l+1)n+k} \rightarrow \beta_{2n+2np+(l+1)n+k+1}\ ]_0$
$[\ \gamma_{2n+2np+(l+1)n+k} \rightarrow \gamma_{2n+2np+(l+1)n+k+1}\ ]_1$
$\left.\begin{array}{l}[\ x_{i,j,2n+2np+(l+1)n+k} \rightarrow x_{i,j,2n+2np+(l+1)n+k+1}\ ]_1 \\ [\ \overline{x}_{i,j,2n+2np+(l+1)n+k} \rightarrow \overline{x}_{i,j,2n+2np+(l+1)n+k+1}\ ]_1 \\ [\ x^*_{i,j,2n+2np+(l+1)n+k} \rightarrow x^*_{i,j,2n+2np+(l+1)n+k+1}\ ]_1\end{array}\right\}$ for $\begin{array}{l}1 \leq i \leq n \\ 1 \leq j \leq p\end{array}$
$[\ c_{j,t} \rightarrow c_{j,t+1}\ ]_2$ for $1 \leq j \leq l+1, 0 \leq t \leq np-2$

Therefore, the following holds

- $\mathcal{C}_{2n+3np}(0) = \{\alpha_{2n+3np}, \beta_{2n+3np}\}$
- In $\mathcal{C}_{2n+3np}$ there are 2^n membranes labelled by 1 such that each of them contains an object γ_{2n+3np}.
- In $\mathcal{C}_{2n+3np}$ there are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ m objects $c_{j,t}$ $(1 \leq j \leq p,\ 0 \leq t \leq np-1)$, that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_p$; and
  - ⋆ $n + np - m$ objects $\#$.

□

5.3 Second checking stage

At this stage, started at configuration $\mathcal{C}_{2n+3np}$, we try to determine the truth assignments that make true the input formula φ, using rules from **5.6**. We are going to divide this stage in two phases. The first one will be devoted to send out all the objects c_j, for $1 \leq j \leq p$ in order to get them ready for the next phase.

Let $k = ln + i$ $(0 \leq l \leq p-1, 1 \leq i \leq n)$, so we can refer to each clause $(l+1)$ when we are doing the verification. Let $m = \sum_{j=1}^{p} m_j$, being m_j the number of objects $c_{j,k}$ in each membrane 2 at step $\mathcal{C}_{2n+3np}$. In this stage, we cannot be sure of how many objects $c_{l+1,k}$ are present at each membrane when $i \neq 0$ [7], so if

[7] That is because objects $c_{j,k}$ do not have to be created consecutively.

we cannot be sure of that, we are going to say that there are $\widetilde{m}_j$ (remember that $\widetilde{m}_j$ is always less than or equal to m_j) objects within membrane 2. We will ignore objects # since they have no effect from here.

Proposition 5. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a) For each k $(1 \leq k \leq np-1)$ at configuration $\mathcal{C}_{2n+3np+k}$ we have the following:
- *$\mathcal{C}_{2n+3np+k}(0) = \{\alpha_{2n+3np+k}, \beta_{2n+3np+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains*
  - ⋆ *an object $\gamma_{2n+3np+k}$; and*
  - ⋆ *m_j objects c_j for $1 \leq j \leq l$ and $m_{l+1} - \widetilde{m}_{l+1}$ objects c_{l+1}*
- *There are 2^n membranes labelled by 2 such that each of them contains $\widetilde{m}_{l+1}$ objects $c_{l+1,t}$ $((p-1)n+1 \leq t \leq np-1)$ and m_j objects $c_{j,t}$ $(l+2 \leq j \leq p, ln+i \leq t \leq np-1)$*

(b) $\mathcal{C}_{2n+4np}(0) = \{\alpha_{2n+4np}, \beta_{2n+4np}\}$, there are 2^n membranes labelled by 1, such that each of them contains m objects c_j $(1 \leq j \leq p)$ and an object γ_{2n+4np}; and 2^n empty membranes labelled by 2.

Proof. (a) is going to be demonstrated by induction on k

- The base case $k = 1$ is trivial because: At configuration $\mathcal{C}_{2n+3np}$ we have: $\mathcal{C}_{2n+3np}(0) = \{\alpha_{2n+3np}, \beta_{2n+3np}\}$ and there exist 2^n membranes labelled by 1 containing an object γ_{2n+3np}; and 2^n membranes labelled by 2 containing m objects $c_{j,t}$ $(1 \leq j \leq k, 0 \leq t \leq np-1)$. Then, configuration $\mathcal{C}_{2n+3np}$ yields configuration $\mathcal{C}_{2n+3np+1}$ by applying the rules:

  $[\ \alpha_{2n+3np} \to \alpha_{2n+3np+1}\ ]_0$
  $[\ \beta_{2n+3np} \to \beta_{2n+3np+1}\ ]_0$
  $[\ \gamma_{2n+3np} \to \gamma_{2n+3np+1}\ ]_1$
  $[c_{j,t} \longrightarrow c_{j,t+1}]_2$, for $1 \leq j \leq p$, $0 \leq k \leq np-2$
  $[c_{1,np-1}]_2 \longrightarrow c_1[\ \ ]_2$

  Thus, $\mathcal{C}_{2n+3np+1}(0) = \{\alpha_{2n+3np+1}, \beta_{2n+3np+1}\}$, and there exist 2^n membranes labelled by 1 containing an object $\gamma_{2n+3np+1}$ and $m_1 - \widetilde{m}_1$ objects c_1 [8]; and 2^n membranes labelled by 2 containing $\widetilde{m}_1$ objects c_1 and m_j objects c_j $(2 \leq j \leq p)$. Hence, the result holds for $k = 1$.
- Supposing, by induction, result is true for k $(1 \leq k \leq np-1)$
  - $\mathcal{C}_{2n+3np+k}(0) = \{\alpha_{2n+3np+k}, \beta_{2n+3np+k}\}$
  - In $\mathcal{C}_{2n+3np+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ an object $\gamma_{2n+3np+k}$; and
    - ⋆ m_j objects c_j for $1 \leq j \leq l$ and $m_{l+1} - \widetilde{m}_{l+1}$ objects c_{l+1}.

[8] That is, if the truth assignment of variable 1 made true clause 1, then an object $c_{1,0}$ were created at $(2n+2np+1)$-th step, and it is going to be sent to the corresponding membrane 1. So, $m_1 - \widetilde{m}_1$ can be 0 or 1 in this step.

- In $\mathcal{C}_{2n+3np+k}$ there are 2^n membranes labelled by 2 such that each of them contains $\widetilde{m}_{l+1}$ objects $c_{l+1,t}$ $((p-1)n+1 \leq t \leq np-1)$ and m_j objects $c_{j,t}$ $(l+2 \leq j \leq p, ln+i \leq t \leq np-1)$.
  Then, configuration $\mathcal{C}_{2n+3np+k}$ yields configuration $\mathcal{C}_{2n+3np+k}$ by applying the rules:
  $[\ \alpha_{2n+3np+k} \rightarrow \alpha_{2n+3np+k+1}\ ]_0$
  $[\ \beta_{2n+3np+k} \rightarrow \beta_{2n+3np+k+1}\ ]_0$
  $[\ \gamma_{2n+3np+k} \rightarrow \gamma_{2n+3np+k+1}\ ]_1$
  $[c_{j,t} \longrightarrow c_{j,t+1}]_2$, for $l+1 \leq j \leq p$, $0 \leq k \leq np-2$
  $[c_{l+1,np-1}]_2 \longrightarrow c_1[\ \ ]_2$
  Therefore, the following holds
- $\mathcal{C}_{2n+3np+k+1}(0) = \{\alpha_{2n+3np+k+1}, \beta_{2n+3np+k+1}\}$
- In $\mathcal{C}_{2n+3np+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ an object $\gamma_{2n+3np+k+1}$; and
  - ⋆ m_j objects c_j for $1 \leq j \leq l$ and $m_{l+1} - \widetilde{m}_{l+1}$ objects c_{l+1}.
- In $\mathcal{C}_{2n+3np+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains $\widetilde{m}_{l+1}$ objects $c_{l+1,t+1}$ $((p-1)n+1 \leq t \leq np-1)$ and m_j objects $c_{j,t+1}$ $(l+2 \leq j \leq p, ln+i \leq t \leq np-1)$.
  Hence, the result holds for $k+1$.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{2n+4np-1}$ holds:
  - $\mathcal{C}_{2n+4np-1}(0) = \{\alpha_{2n+4np-1}, \beta_{2n+4np-1}\}$
  - In $\mathcal{C}_{2n+4np-1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ an object $\gamma_{2n+4np-1}$; and
    - ⋆ m_j objects c_j for $1 \leq j \leq p-1$ and $m_p - \widetilde{m}_p$ [9] objects c_p.
  - In $\mathcal{C}_{2n+4np-1}$ there are 2^n membranes labelled by 2 such that each of them contains $\widetilde{m}_p$ objects $c_{p,np}$.
    Then, configuration $\mathcal{C}_{2n+4np-1}$ yields configuration $\mathcal{C}_{2n+4np}$ by applying the rules:
    $[\ \alpha_{2n+4np-1} \rightarrow \alpha_{2n+4np}\ ]_0$
    $[\ \beta_{2n+4np-1} \rightarrow \beta_{2n+4np}\ ]_0$
    $[\ \gamma_{2n+4np-1} \rightarrow \gamma_{2n+4np}\ ]_1$
    $[c_{p,np}]_2 \longrightarrow c_p[\ \ ]_2$
    Then, we have $\mathcal{C}_{2n+4np}(0) = \{\alpha_{2n+4np}, \beta_{2n+4np}\}$, and there exist 2^n membranes labelled by 1 containing an object γ_{2n+4np} and m objects c_j $(1 \leq j \leq p)$; and there exist 2^n empty membranes labelled by 2.

□

When objects c_j are within the membranes labelled by 1, we can start to check if all the clauses of the input formula φ are satisfied by any truth assignment. As we use objects c_j to denote that clause C_j has been satisfied by some variable, it

[9] In this case, $\widetilde{m}_p$ can only take two values: 0 or 1.

can be possible that some c_j are missing, that is, that for some j, $1 \leq j \leq p$, c_j does not appear in any membrane labelled by 1 in $\mathcal{C}_{2n+4np}$. Let $\widetilde{j}$ be the index j [10] of that clause. It is going to take $2p$ steps.

Proposition 6. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a_0) *For each $2k+1$ ($0 \leq k \leq p-1$) at configuration $\mathcal{C}_{2n+4np+2k+1}$ we have the following:*
- *$\mathcal{C}_{2n+4np+2k+1}(0) = \{\alpha_{2n+4np+2k+1}, \beta_{2n+4np+2k+1}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains*
  - ⋆ *an object γ_{2n+4np} or $d_{\widetilde{j}-1}$ (respectively, an object d_k) if the corresponding truth assignment does not make true (resp., makes true) the clause C_1 or C_j ($2 \leq j \leq p$) (resp., the first k clauses); and*
  - ⋆ *$m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k+1)$ and m_j objects c_j for $min(\widetilde{j}, k+2) \leq j \leq p$.*
- *There are 2^n membranes labelled by 2 such that each of them contains an object d_{k+1} if and only if the truth assignment associated to the branch makes true the first $k+1$ clauses.*

(a_1) *For each $2k$ ($1 \leq k \leq p-1$) at configuration $\mathcal{C}_{2n+4np+2k}$ we have the following:*
- *$\mathcal{C}_{2n+4np+2k}(0) = \{\alpha_{2n+4np+2k}, \beta_{2n+4np+2k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains*
  - ⋆ *an object γ_{2n+4np} or $d_{\widetilde{j}-1}$ if the corresponding truth assignment does not make true the clause C_1 or C_j ($2 \leq j \leq p$); and*
  - ⋆ *$m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k)$ and m_j objects c_j for $min(\widetilde{j}, k+1) \leq j \leq p$.*
- *There are 2^n empty membranes labelled by 2.*

(b) *$\mathcal{C}_{2n+4np+2p}(0) = \{\alpha_{2n+4np+2p}, \beta_{2n+4np+2p}\}$, and in $\mathcal{C}_{2n+4np+2p}$ there are 2^n membranes labelled by 1, such that each of them contains an object d_p if and only if the corresponding truth assignment makes true the input formula φ ($d_{\widetilde{j}-1}$ otherwise), $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, p+1)$ and m_j objects c_j for $min(\widetilde{j}, p+1) \leq j \leq p$; and 2^n empty membranes labelled by 2.*

Proof. (a) is going to be demonstrated by induction on k

- The base case $k = 1$ is trivial because:

(a_0) at configuration $\mathcal{C}_{2n+4np}$ we have: $\mathcal{C}_{2n+4np}(0) = \{\alpha_{2n+4np}, \beta_{2n+4np}\}$ and there exist 2^n membranes labelled by 1 containing an object γ_{2n+4np} and m objects c_j ($1 \leq j \leq p$); and there exist 2^n empty membranes labelled by 2. Then, configuration $\mathcal{C}_{2n+4np}$ yields configuration $\mathcal{C}_{2n+4np+1}$ by applying the rules:

$$[\ \alpha_{2n+4np} \to \alpha_{2n+4np+1}\ ]_0$$
$$[\ \beta_{2n+4np} \to \beta_{2n+4np+1}\ ]_0$$
$$\gamma_{4np+2n}\ c_1[\ \ ]_2 \longrightarrow [\ d_1\ ]_2$$

[10] If $\widetilde{j}$ is not defined, we are going to suposse that it is equal to $p+1$.

(a_1) at $\mathcal{C}_{2n+4np+1}(0) = \{\alpha_{2n+4np+1}, \beta_{2n+4np+1}\}$ and there exist 2^n membranes labelled by 1 containing an object γ_{2n+4np} if and only if there were no objects c_1 at configuration $\mathcal{C}_{2n+4np}$, $m_1 - 1$ (respectively, m_1) objects c_1 if there was any object c_j in this membrane in the previous configuration (resp., m_1) and m_j objects c_j for $2 \leq j \leq p$; and 2^n membranes labelled by 2 containing an object d_1 if and only if there was at least one object c_1 within membrane labelled by 1 at configuration $\mathcal{C}_{2n+4np}$. Then, the configuration $\mathcal{C}_{2n+4np+1}$ yields configuration $\mathcal{C}_{2n+4np+2}$ by applying the rules:

$[\ \alpha_{2n+4np+1} \rightarrow \alpha_{2n+4np+2}\ ]_0$
$[\ \beta_{2n+4np+1} \rightarrow \beta_{2n+4np+2}\ ]_0$
$[\ d_1\ ]_2 \longrightarrow\ d_1\ [\ \ ]_2$

Thus, $\mathcal{C}_{2n+4np+2}(0) = \{\alpha_{2n+4np+2}, \beta_{2n+4np+2}\}$, and there exist 2^n membranes labelled by 1 containing an object d_1 (respectively, γ_{2n+4np}) if the corresponding truth assignment makes true (resp., doesn't make true) clause C_1, $m_1 - 1$ (resp., m_1) objects c_1 and m_j objects c_j for $1 \leq j \leq p$; and there exist 2^n empty membranes labelled by 2. Hence, the result holds for $k = 1$.

- Supposing, by induction, result is true for k $(0 \leq k \leq p-1)$
  - $\mathcal{C}_{2n+4np+2k}(0) = \{\alpha_{2n+4np+2k}, \beta_{2n+4np+2k}\}$
  - In $\mathcal{C}_{2n+4np+2k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ an object γ_{2n+4np} or $d_{\widetilde{j-1}}$ (respectively, an object d_k) if the corresponding truth assignment does not make true (resp., makes true) the clause C_1 or C_j $(2 \leq j \leq p)$ (resp., the first k clauses); and
    - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k+1)$ and m_j objects c_j for $min(\widetilde{j}, k+2) \leq j \leq p$.
  - In $\mathcal{C}_{2n+4np+2k}$ there are 2^n empty membranes labelled by 2.

  Then, configuration $\mathcal{C}_{2n+4np+2k}$ yields configuration $\mathcal{C}_{2n+4np+2k+1}$ by applying the rules:

  $[\ \alpha_{2n+4np+2k} \rightarrow \alpha_{2n+4np+2k+1}\ ]_0$
  $[\ \beta_{2n+4np+2k} \rightarrow \beta_{2n+4np+2k+1}\ ]_0$
  $d_k\ c_{k+1}[\ \ ]_2 \longrightarrow\ [\ d_{k+1}\ ]_2$

  Therefore, the following holds
  - $\mathcal{C}_{2n+4np+2k+1}(0) = \{\alpha_{2n+4np+2k+1}, \beta_{2n+4np+2k+1}\}$
  - In $\mathcal{C}_{2n+4np+2k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ an object γ_{2n+4np} or $d_{\widetilde{j-1}}$ if the corresponding truth assignment does not make true the clause C_1 or C_j $(2 \leq j \leq p)$; and
    - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k)$ and m_j objects c_j for $min(\widetilde{j}, k+1) \leq j \leq p$.
  - In $\mathcal{C}_{2n+4np+2k+1}$ there are 2^n membranes labelled by 2 such that each of them contains an object d_{k+1} if and only if the corresponding truth assignment makes true the first $k+1$ clauses.

Then, configuration $\mathcal{C}_{2n+4np+2k+1}$ yields $\mathcal{C}_{2n+4np+2k+2}$ by applying the rules:

$[\ \alpha_{2n+4np+2k+1} \rightarrow \alpha_{2n+4np+2k+2}\ ]_0$
$[\ \beta_{2n+4np+2k+1} \rightarrow \beta_{2n+4np+2k+2}\ ]_0$
$[\ d_{k+1}\ ]_2 \longrightarrow\ d_{k+1}\ [\ \ ]_2$

Therefore, the following holds

- $\mathcal{C}_{2n+4np+2k+2}(0) = \{\alpha_{2n+4np+2k+2}, \beta_{2n+4np+2k+2}\}$
- In $\mathcal{C}_{2n+4np+2k+2}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ an object γ_{2n+4np} or $d_{\widetilde{j}-1}$ (respectively, an object d_{k+1}) if the corresponding truth assignment does not make true (resp., makes true) the clause C_1 or C_j $(2 \leq j \leq p)$ (resp., the first $k+1$ clauses); and
  - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k+2)$ and m_j objects c_j for $min(\widetilde{j}, k+3) \leq j \leq p$.
- In $\mathcal{C}_{2n+4np+2k+2}$ there are 2^n empty membranes labelled by 2.

Hence, the result holds for $k+1$.

- In order to prove (b) it is enough to notice that, on the one han, from (a) configuration $\mathcal{C}_{2n+4np+2p-1}$ holds:
  - $\mathcal{C}_{2n+4np+2p-1}(0) = \{\alpha_{2n+4np+2p-1}, \beta_{2n+4np+2p-1}\}$
  - In $\mathcal{C}_{2n+4np+2p-1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - ⋆ an object γ_{2n+4np} or $d_{\widetilde{j}-1}$ if the corresponding truth assignment does not make true the clause C_1 or C_j $(2 \leq j \leq p)$; and
    - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, p)$ and m_j objects c_j for $min(\widetilde{j}, p+1) \leq j \leq p$.
  - In $\mathcal{C}_{2n+4np+2p-1}$ there are 2^n membranes labelled by 2 such that each of them contains an object d_p if and only if the corresponding truth assignment makes true the input formula φ.

  Then, configuration $\mathcal{C}_{2n+4np+2p-1}$ yields configuration $\mathcal{C}_{2n+4np+2p}$ by applying the rules:

  $[\ \alpha_{2n+4np+2p-1} \rightarrow \alpha_{2n+4np+2p}\ ]_0$
  $[\ \beta_{2n+4np+2p-1} \rightarrow \beta_{2n+4np+2p}\ ]_0$
  $[\ d_p\ ]_2 \longrightarrow\ d_p\ [\ \ ]_2$

  Then, we have $\mathcal{C}_{2n+4np+2p}(0) = \{\alpha_{2n+4np+2p}, \beta_{2n+4np+2p}\}$, and there exist 2^n membranes labelled by 1 containing an object γ_{2n+4np} or $d_{\widetilde{j}-1}$ (respectively, an object d_p) if the corresponding truth assignment does not make true (resp., makes true) the clause C_1 or C_j $(2 \leq j \leq p)$ (resp., the input formula φ), $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, p+1)$ and m_j objects c_j for $min(\widetilde{j}, p+1) \leq j \leq p$; and there exist 2^n empty membranes labelled by 2.

□

5.4 Output stage

The output phase starts at the $(2n + 4np + 2p)$-th step, and takes exactly four steps when there is an affirmative answer and five steps when there is a negative one. Rules from **5.7** are devoted to compute this stage.

- *Affirmative answer*: In this case, at configuration $\mathcal{C}_{2n+4np+2p}$, in some membrane 1 there is an object d_p. By applying the rule $[\ d_p\ ]_1 \longrightarrow d_p[\ \ ]_1$ (at the same time that $[\ \alpha_{2n+4np+2p} \rightarrow \alpha_{2n+4np+2p+1}\ ]_0$ and $[\ \beta_{2n+4np+2p} \rightarrow \beta_{2n+4np+2p+1}\ ]_0$ are executed), an object d_p is produced in membrane 0. Then by applying the rules $\alpha_{4np+2n+2p+1}\ d_p[\ \ ]_1 \longrightarrow [\ \texttt{yes}\ ]_1$ and $[\ \beta_{2n+4np+2p+1} \rightarrow \beta_{2n+4np+2p+2}\ ]_0$, an object `yes` is produced in some membrane labelled by 1 (only in one such membrane). At the next step, an object `yes` will appear at membrane labelled by 0 of the configuration $\mathcal{C}_{2n+4np+2p+3}$ by the application of the rule $[\ \texttt{yes}\ ]_1 \longrightarrow \texttt{yes}[\ \ ]_1$. Let us note that object $\beta_{2n+4np+2p+2}$ cannot interact with any object α. Finally, at computation step $2n + 4np + 2p + 4$ an object `yes` is released to environment by the application of the rule $[\ \texttt{yes}\ ]_0 \longrightarrow \texttt{yes}[\ \ ]_0$ and the computation halts.
- *Negative answer*: In this case, at configuration $\mathcal{C}_{2n+4np+2p}$, there are no membranes labelled by 1 that contains an object d_p, so the only rules executed are $[\ \alpha_{2n+4np+2p} \rightarrow \alpha_{2n+4np+2p+1}\ ]_0$ and $[\ \beta_{2n+4np+2p} \rightarrow \beta_{2n+4np+2p+1}\ ]_0$. Rule $[\ \beta_{2n+4np+2p+1} \rightarrow \beta_{2n+4np+2p+2}\ ]_0$ is executed in the next step. Thus, at configuration $\mathcal{C}_{2n+4np+2p+2}$ in membrane labelled by 0 we execute have a copy of object $\alpha_{2n+4np+2p+1}$ and a copy of object $\beta_{2n+4np+2p+2}$. By applying the rule $\alpha_{4np+2n+2p+1}\ \beta_{4np+2n+2p+2}[\ \ ]_1 \longrightarrow [\ \texttt{no}\ ]_1$, an object `no` is produced in only one membrane labelled by 1 (nondeterministically chosen). At the next step, this object `no` will move into membrane labelled by 0 by the application of the rule $[\ \texttt{no}\ ]_1 \longrightarrow \texttt{no}[\ \ ]_1$. Finally, at configuration $\mathcal{C}_{2n+4np+2p+5}$ an object `no` is released to the environment when rule $[\ \texttt{no}\ ]_0 \longrightarrow \texttt{no}[\ \ ]_0$, and then the computation halts.

5.5 Result

Theorem 1. $\texttt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)}$.

Proof. The family $\mathbf{\Pi}$ of P systems previously constructed verifies the following:

(a) The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines because for each $n, p \in \mathbb{N}$, the rules of $\Pi(\langle n, p\rangle)$ of the family are recursively defined from $n, p \in \mathbb{N}$, and the amount of resources needed to build an element of the family is of a polynomial order in n and p, as shown below:
   - Size of the alphabet: $\frac{15n^2p^2}{2}+6n^2p+3n^2+2np^2+\frac{35np}{2}+8n+7p+9 \in \Theta(n^2p^2)$.
   - Initial number of membranes: $3 \in \Theta(1)$.
   - Initial number of objects in membranes: $3np + n + 3 \in \Theta(np)$.
   - Number of rules: $\frac{15n^2p^2}{2} + 8n^2p + 4n^2 + \frac{41np}{2} + 5n + 5p + 11 \in \Theta(n^2p^2)$.

– Maximal number of objects involved in any rule: $3 \in \Theta(1)$.

(b) The family $\mathbf{\Pi}$ is polynomially bounded with regard to (SAT, cod, s): indeed for each instance φ of the SAT problem, any computation of the system $\mathbf{\Pi}(s(\varphi))$ with input multiset $cod(\varphi)$ takes at most $2n + 4np + 2p + 5$ computation steps.

(e) The family $\mathbf{\Pi}$ is sound with regard to (SAT, cod, s): indeed for each instance φ of the SAT problem, if the computation of $\mathbf{\Pi}(s(\varphi)) + cod(\varphi)$ is an accepting computation, then φ is satisfiable.

(f) The family $\mathbf{\Pi}$ is complete with regard to (SAT, cod, s): indeed, for each instance φ of the SAT problem such that φ is satisfiable, any computation of $\mathbf{\Pi}(s(\varphi)) + cod(\varphi)$ is an accepting computation.

Therefore, the family $\mathbf{\Pi}$ of P systems previously constructed solves the SAT problem in polynomial time and in a uniform way.

Corollary 1. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)}$.

Proof. It suffices to notice that SAT problem is a **NP**-complete problem, SAT $\in \mathbf{PMC}_{\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)}$, and the complexity class $\mathbf{PMC}_{\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)}$ is closed under polynomial-time reduction and under complement.

6 Conclusions

From a computational complexity point of view and assuming that $\mathbf{P} \neq \mathbf{NP}$, dissolution rules play a crucial role in classical polarizationless P systems with active membranes where there is no cooperation, no changing labels neither priorities. In that framework, **PSPACE**-complete problems can be solved in polynomial time when dissolution rules and division for elementary and non-elementary membranes are permitted. However, dissolution rules and division rules for non-elementary membranes can be replaced by minimal cooperation (the left-hand side of the rules has at most two objects) and minimal production (the right-hand side of the rules has at most two objects) in object evolution rules in order to obtain the computational efficiency [11].

In this paper, the ingredient of minimal cooperation and minimal production in object evolution rules is replaced by minimal cooperation and minimal production in send-in communication rules but we have need to use division for non-elementary membranes. The new systems considered are able to efficiently solve computational hard problems even by considering *simple object evolution rules*, that is, these kind of rules only produce one object. An analogous result can be obtained if minimal cooperation and minimal production are considered only for send-out rules, instead of send-in rules ([12]).

The case where only elementary division is allowed, while keeping the restriction that minimal cooperation and minimal production are used in communication rules of the same direction (only *in* or only *out*) remains as future work, as well as the case where division rules are replaced by separation rules.

What about the class $\mathcal{SAM}^0(+e_s, mcmp_{in}, -d, +n)$? That is, what happens if we revisit the framework studied in this paper but replacing division rules by separation rules? We can adapt the reasoning used in the proof of $\mathbf{P} = \mathbf{PMC}_{\mathcal{SAM}^0_{bmc}(-d,-n)}$ (see [10]), and we can prove that by using families of recognizer membrane systems belonging to this class, only problems in class **P** can be solved in polynomial time.

Acknowledgements

This work was partially supported by Grant numbers 61472328 and 61320106005 of the National Natural Science Foundation of China.

References

1. A. Alhazov, L. Pan. Polarizationless P systems with active membranes. *Grammars*, **7** (2004), 141-159.
2. A. Alhazov, L. Pan, Gh. Păun. Trading polarizations for labels in P systems with active membranes. *Acta Informaticae*, **41**, 2-3 (2004), 111-144.
3. T.H. Cormen, C.E. Leiserson, R.L. Rivest. *An Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1994.
4. M.R. Garey, D.S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
5. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, F.J. Romero–Campero. On the power of dissolution in P systems with active membranes. In R. Freund, Gh. Păun, Gr. Rozenberg, A. Salomaa (eds.). *Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, Lecture Notes in Computer Science*, **3850** (2006), 224–240.
6. Gh. Păun. P systems with active membranes: Attacking **NP**–complete problems, *Journal of Automata, Languages and Combinatorics*, **6** (2001), 75–90. A preliminary version in *Centre for Discrete Mathematics and Theoretical Computer Science Research Reports Series*, CDMTCS-102, May 1999.
7. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
8. P. Sosík, A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, **73** (2007), 137152.
9. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Polarizationless P systems with active membranes: Computational complexity aspects. *Journal of Automata, Languages and Combinatorics*, **21**, 1-2 (2016), 107123
10. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núñez, M.J. Pérez-Jiménez. Minimal cooperation in polarizationless P systems with active membranes. In C. Graciani, Gh. Păun, D. Orellana-Martn, A. Riscos-Nez, L. Valencia-Cabrera (eds.) *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 1-5 February, 2016, Sevilla, Spain, Fénix Editora, pp. 327-356.

11. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Reaching efficiency through collaboration in membrane systems: dissolution, polarization and cooperation. *Theoretical Computer Science*, in press, 2017.
12. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Restricted polarizationless P systems with active membranes: minimal cooperation only outwards. In this volume, 2017 (manuscript).
13. C. Zandron, C. Ferretti, G. Mauri. Solving **NP**-complete problems using P systems. In I. Antoniou, C.S. Calude, M.J. Dinneen (eds.) *Unconventional Models of Computation, UMC'2K*, Springer, London, 2000, pp. 153-164.

Restricted Polarizationless P Systems with Active Membranes: Minimal Cooperation Only Outwards

Luis Valencia-Cabrera, David Orellana-Martín, Miguel Á. Martínez-del-Amor, Agustín Riscos-Núñez, Mario J. Pérez-Jiménez

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

Summary. Membrane computing is a computing paradigm providing a class of distributed parallel computing devices of a biochemical type whose process units represent biological membranes. In the cell-like basic model, a hierarchical membrane structure formally described by a rooted tree is considered. It is well known that families of such systems where the number of membranes can only decrease during a computation (for instance by dissolving membranes), can only solve in polynomial time problems in class **P**. *P systems with active membranes* is a variant where membranes play a central role in their dynamics. In the seminal version, membranes have an electrical polarization (positive, negative, or neutral) associated in any instant, and besides being dissolved, they can also replicate by using division rules. These systems are computationally universal, that is, equivalent in power to deterministic Turing machines, and computationally efficient, that is, able to solve computationally hard problems in polynomial time. If polarizations in membranes are removed and dissolution rules are forbidden, then only problems in class **P** can be solved in polynomial time by these systems (even in the case when division rules for non-elementary membranes are permitted). In that framework it has been shown that by considering minimal cooperation (left-hand side of such rules consists of at most two symbols) and minimal production (only one object is produced by the application of such rules) in object evolution rules, such systems provide efficient solutions to **NP**–complete problems. In this paper, minimal cooperation and minimal production in communication rules instead of object evolution rules is studied, and the computational efficiency of these systems is obtained in the case where division rules for non-elementary membranes are permitted.

Key words: Membrane Computing, polarizationless P systems with active membranes, cooperative rules, the **P** versus **NP** problem, SAT problem.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing providing distributed parallel and non-deterministic computing models whose computational devices are called *membrane systems* having units processor called *compartments*. This computing paradigm is inspired by some basic biological features, by the structure and functioning of the living cells, as well as from the cooperation of cells in tissues, organs, and organisms. Celllike membrane systems use the biological membranes arranged hierarchically, inspired from the structure of the cell.

In Membrane Computing, some variants capture the fact that membranes are not at all passive from a biochemistry view, for instance, the passing of a chemical compound through a membrane is often done by a direct interaction with the membrane itself. Some variants of P systems where the central role in their dynamics is played by the membranes have been considered. In these models, the membranes not only directly mediate the evolution and the communication of objects, but they can replicate themselves by means of a division process. Inspired by these features, *P systems with active membranes* [6] were introduced, based on processing multisets by means of non-cooperative rewriting rules, that is, rules where its left-hand side has at most only one object. Specifically, objects evolve inside membranes which can communicate between each other, can dissolve, and moreover (inspired by cellular mitosis process) can replicate by means of division rules. It is assumed that each membrane has associated an electrical polarization in any instant, one of the three possible: positive, negative, or neutral.

P systems with active membranes are computationally complete, that is, any recursively enumerable set of vectors of natural numbers (in particular, each recursively enumerable set of natural numbers) can be generated by such a system [6]. Hence, they are equivalent in power to deterministic Turing machines.

What about the computational efficiency of P systems with active membranes? The key is certainly in the use of division rules, as we can deduce from the so-called *Milano theorem* [13]: *A deterministic P system with active membranes but without membrane division can be simulated by a deterministic Turing machine with a polynomial slowdown.*

However, P systems with active membranes which make use of division rules have the ability to provide efficient solutions to computationally hard problems, by making use of an exponential workspace created in a polynomial time. Specifically, **NP**-complete problems can be solved in polynomial time by families of P systems with active membranes, without dissolution rules and which use division rules only for elementary membranes [6]. Moreover, the class of decision problems which can be solved by families of P systems with active membranes with dissolution rules and which use division for elementary and non-elementary membranes is equal to **PSPACE** [8]. Consequently, the usual framework of P systems with active membranes and electrical polarizations for solving decision problems seems to be too powerful from the computational complexity point of view.

With respect to the computational efficiency, in the classical framework of P system with active membranes, dissolution rules play an "innocent" role as well as

division for non-elementary membranes. However, if electrical charges are removed then these kind of rules come to play a relevant role. Specifically, P systems with active membranes and without electrical charges were initially studied in [1, 2] by replacing electrical charges by the ability to change the label of the membranes. In this paper, polarizationless P systems with active membranes where labels of membranes keep unchanged by the application of rules, are considered. In this new framework, if dissolution rules are forbidden then only problems in class **P** can be solved in an efficient way, even in the case that division for non-elementary membranes are permitted [5]. Is the class of polarizationless P systems with active membranes, with dissolution but using only division rules for elementary membranes computationally efficient? If $\mathbf{P} \neq \mathbf{NP}$, which is at all expected, then it is an open question, so-called *Păun's conjecture*.

In the seminal paper where P systems with active membranes were introduced, Gh. Păun says that "*working with non-cooperative rules is natural from a mathematical point of view but from a biochemical point of view this is not only non-necessary, but also non-realistic: most of the chemical reactions involve two or more than two chemical compounds* (*and also produce two or more than two compounds*)". In this context, a restricted cooperation has been considered in the classical framework of polarizationless P systems with active membranes. Specifically, minimal cooperation (the left-hand side and the right-hand side of any rules have, at most, two objects) in object evolution rules, has been previously studied from a computational complexity point of view. A polynomial-time solution to the `SAT` problem by means of families of polarizationless P systems with active membranes, with minimal cooperation in object evolution rules, has been provided [9]. Recently, this result has been improved by considering minimal cooperation in object evolution rules with and additional restriction: the right-hand side of any rules has only one object (called *minimal cooperation and minimal production*) [11]. A relevant fact in these results is the following: dissolution rules and division rules for non-elementary membranes are not necessary to reach the computational efficiency.

In this paper the role of minimal cooperation and minimal production in communication rules instead of object evolution rules, is studied from a complexity point of view. Specifically, by using families of membrane systems which use these syntactical ingredients, a polynomial-time solution to the `SAT` problem is provided but allowing division rules for non-elementary membranes.

The paper is structured as follows. First, some basic notions are recalled and the terminology and notation to be used in the paper is presented. Then, Section 3 introduces the model that will be investigated in this paper: polarizationless P systems with active membranes, with minimal cooperation and minimal production in their communication rules. Section 4 contains the main result of this paper, showing that these systems are capable of solving an **NP**-complete problem in an *efficient* way. Finally, the paper concludes with some final remarks and ideas for future work.

2 Preliminaries

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

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

2.1 Graphs and trees

Let us recall some notions related with graph theory (see [3] for details). An *undirected graph* is an ordered pair (V, E) where V is a set whose elements are called nodes or vertices and $E = \{\{x, y\} \mid x \in V, y \in V, x \neq y\}$ whose elements are called *edges*. A *path* of length $k \geq 1$ from a node u to a node v in a graph (V, E) is a finite sequence $(x_0, x_1, \ldots, x_k)$ of nodes such that $x_0 = u$, $x_k = v$ and $\{x_i, x_{i+1}\} \in E$. If $k \geq 2$ and $x_0 = x_k$ then we say that the path is a *cycle* of the graph. A graph with no cycle is said to be *acyclic*. An undirected graph is *connected* if there exist paths between every pair of nodes.

A *rooted tree* is a a connected, acyclic, undirected graph in which one of the vertices (called *the root of the tree*) is distinguished from the others. Given a node x (different from the root), if the last edge on the (unique) path from the root of the tree to the node x is $\{x, y\}$ (in this case, $x \neq y$), then y is **the** *parent* of node x and x is **a** *child* of node y. The root is the only node in the tree with no parent. A node with no children is called a *leaf*.

2.2 The Cantor pairing function

The Cantor pairing function encodes pairs of natural numbers by single natural numbers, and it is defined as follows: for each $n, p \in \mathbb{N}$

$$\langle n, p \rangle = \frac{(n+p)(n+p+1)}{2} + n$$

The Cantor pairing function is a primitive recursive function and bijective from $\mathbb{N} \times \mathbb{N}$ onto $\mathbb{N}$. Then, for each $t \in \mathbb{N}$ there exist unique natural numbers $n, p \in \mathbb{N}$ such that $t = \langle n, p \rangle$.

2.3 Decision problems and languages

A decision problem X is an ordered pair (I_X, θ_X), where I_X is a language over a finite alphabet Σ_X and θ_X is a total Boolean function over I_X. The elements of I_X are called *instances* of the problem X. Each decision problem X has associated a language L_X over the alphabet Σ_X as follows: $L_X = \{u \in {\Sigma_X}^* \mid \theta_X(u) = 1\}$. Conversely, every language L over an alphabet Σ has associated a decision problem $X_L = (I_{X_L}, \theta_{X_L})$ as follows: $I_{X_L} = \Sigma^*$ and $\theta_{X_L}(u) = 1$ if and only if $u \in L$. Therefore, given a decision problem X we have $X_{L_X} = X$, and given a language L over an alphabet Σ we have $L_{X_L} = L$. Then, solving a decision problem can be expressed equivalently as the task of recognizing the language associated with it.

2.4 Recognizer membrane systems

Recognizer membrane systems were introduced in [7] and they provide a natural framework to solve decision problems. This kind of systems are characterized by the following features: (a) the working alphabet Γ has two distinguished objects `yes` and `no`; (b) there exists an input membrane and an input alphabet Σ strictly contained in Γ; (c) the initial contents of the membranes are multisets over $\Gamma \setminus \Sigma$; (d) all computations halt; and (e) for each computation, either object `yes` or object `no` (but not both) must have been released into the environment, and only at the last step of the computation.

Given a recognizer membrane system, Π, for each multiset m over the input alphabet Σ we denote by $\Pi + m$ the membrane system Π with input multiset m, that is in the initial configuration of that system, the multiset m is added to the initial content of the input membrane. Thus, in a recognizer membrane system, Π, there exists an initial configuration associated with each multiset $m \in M_f(\Sigma)$.

3 Minimal cooperation and minimal production in communication rules

Definition 1. *A polarizationless P system with active membranes, with simple object evolution rules, without dissolution, with division rules for elementary and non-elementary membranes, and which makes use of minimal cooperation and minimal production in send-out communication rules, is a tuple*

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

where:

- Γ *is a finite alphabet whose elements are called objects and contains two distinguished objects* `yes` *and* `no`*.*
- $\Sigma \subsetneq \Gamma$ *is the input alphabet.*

- H *is a finite alphabet such that* $H \cap \Gamma = \emptyset$ *whose elements are called labels.*
- $q \geq 1$ *is the degree of the system.*
- μ *is a labelled rooted tree (called membrane structure) consisting of* q *nodes injectively labelled by elements of* H *(the root of* μ *is labelled by* r_μ*).*
- $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are multisets over* $\Gamma \setminus \Sigma$*.*
- $\mathcal{R}$ *is a finite set of rules, of the following forms:*

  (a_0) $[\, a \to b \,]_h$*, where* $h \in H$*,* $a, b \in \Gamma$*,* $u \in M_f(\Gamma)$ (simple object evolution rules).

  (b_0) $a\,[\ \ ]_h \to [\, b \,]_h$*, where* $h \in H \setminus \{r_\mu\}$*,* $a, b, c \in \Gamma$ (send–in communication rules).

  (c_0) $[\, a\, b \,]_h \to c\,[\ \ ]_h$*, where* $h \in H$*,* $a, b \in \Gamma$ (send–out communication rules with minimal cooperation and minimal production).

  (d_0) $[\, a \,]_h \to b$*, where* $h \in H \setminus \{i_{out}, r_\mu\}$*,* $a, b \in \Gamma$ (dissolution rules).

  (e_0) $[\, a \,]_h \to [\, b \,]_h\, [\, c \,]_h$*, where* $h \in H \setminus \{i_{out}, r_\mu\}$*,* $a, b, c \in \Gamma$ *and* h *is the label of an elementary membrane* μ (division rules for elementary membranes).

  (f_0) $[\,[\ ]_{h_1}[\ ]_{h_2}\,]_{h_0} \to [\,[\ ]_{h_1}\,]_{h_0}\ [\,[\ ]_{h_2}\,]_{h_0}$*, where* $h_0, h_1, h_2 \in H$ *and* $h_0 \neq r_\mu$ (division rules for non-elementary membranes).
- $i_{in} \in H$*,* $i_{out} \in H \cup \{env\}$ *(if* $i_{out} \in H$ *then* i_{out} *is the label of a leaf of* μ*).*

In a similar way is defined the concept of "*polarizationless P system with active membranes, with simple object evolution rules, without dissolution, with division rules for elementary and non-elementary membranes, and which makes use of minimal cooperation and minimal production in send-in communication rules* ". The only difference concerns rules of type (b_0) and (c_0). In this case are, respectively:

(b_0') $a\,b\,[\ \ ]_h \to [\, c \,]_h$ for $h \in H \setminus \{r_\mu\}$, $a, b \in \Gamma$ *(send–in communication rules with minimal cooperation and minimal production).*

(c_0') $[\, a \,]_h \to b\,[\ \ ]_h$ for $h \in H$, $a, b, c \in \Gamma$ *(send–out communication rules).*

The semantics of this kind of P systems follows the usual principles of P systems with active membranes [6].

We denote by $\mathcal{DAM}^0(+e_s, mcmp_{out}, -d, +n)$ (respectively, $\mathcal{DAM}^0(+e_s, mcmp_{in}, -d, +n)$) the class of all recognizer polarizationless P system with active membranes, with simple object evolution rules, without dissolution, with division rules for elementary and non-elementary membranes, which make use of minimal cooperation and minimal production in send-out (respectively, send-in) communication rules.

4 Solving SAT in $\mathcal{DAM}^0(+e_s, mcmp_{out}, -d, +n)$

In this section, a polynomial-time solution to SAT problem, is explicitly given in the framework of recognizer polarizationless P systems with active membranes

with simple object evolution rules, without dissolution and with division rules for elementary and non-elementary membranes which make use of minimal cooperation and minimal production in send-in communication rules. For that, a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of recognizer P systems from $\mathcal{DAM}^0(+e_s, mcmp_{out}, -d, +n)$ will be presented.

4.1 Description of a solution to SAT problem in $\mathcal{DAM}^0(+e_s, mcmp_{out}, -d, +n)$

For each $n, p \in \mathbb{N}$, we consider the recognizer P system

$$\Pi(\langle n, p\rangle) = (\Gamma, \Sigma, H, \mu, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in}, i_{out})$$

from $\mathcal{DAM}^0(+e_s, mcmp_{out}, -d, +n)$, defined as follows:

(1) Working alphabet:

$$\begin{aligned}\Gamma = \Sigma \cup \{\texttt{yes}, \texttt{no}, \#\} \cup \{a_{i,k} \mid 1 \le i \le n \wedge 1 \le k \le 2i-1\} \cup \\ \{\alpha_k \mid 0 \le k \le 4np+n+2p\} \cup \{\beta_k \mid 0 \le k \le 4np+n+2p+1\} \cup \\ \{\gamma_k \mid 0 \le k \le 4np+n\} \cup \\ \{t_{i,k}, f_{i,k} \mid 1 \le i \le n \wedge 2i-1 \le k \le 2n+2p-1\} \cup \\ \{T_i, F_i \mid 1 \le i \le n\} \cup \{c_j \mid 1 \le j \le p\} \cup \\ \{c_{j,k} \mid 1 \le j \le p \wedge 0 \le k \le np-1\} \cup \{d_j \mid 1 \le j \le p\} \cup \\ \{x_{i,j,k}, \overline{x}_{i,j,k}, x^*_{i,j,k} \mid 1 \le i \le n \wedge 1 \le j \le p \wedge \\ 1 \le k \le n+2np+n(j-1)+(i-1)\}\end{aligned}$$

where the input alphabet is $\Sigma = \{x_{i,j,0}, \overline{x}_{i,j,0}, x^*_{i,j,0} \mid 1 \le i \le n \wedge 1 \le j \le p\}$;

(2) $H = \{0, 1, 2\}$;

(3) Membrane structure: $\mu = [\,[\,[\quad]_2\,]_1\,]_0$, that is, $\mu = (V, E)$ where $V = \{0, 1, 2\}$ and $E = \{(0, 1)(1, 2)\}$

(4) Initial multisets: $\mathcal{M}_0 = \{\alpha_0, \beta_0\}$, $\mathcal{M}_1 = \emptyset$, $\mathcal{M}_2 = \{\gamma_0\} \cup \{a_{i,1}, T_i^p, F_i^p \mid 1 \le i \le n\}$.

(5) The set of rules $\mathcal{R}$ consists of the following rules:

5.1 Counters for synchronize the answer of the system.

$$\begin{aligned}&[\,\alpha_k \longrightarrow \alpha_{k+1}\,]_0, \text{ for } 0 \le k \le 4np+n+2p-1\\ &[\,\beta_k \longrightarrow \beta_{k+1}\,]_0, \text{ for } 0 \le k \le 4np+n+2p\\ &[\,\gamma_k \longrightarrow \gamma_{k+1}\,]_2, \text{ for } 0 \le k \le 4np+n-1\end{aligned}$$

5.2 Rules to generate 2^n membranes labelled by 1 and 2^n membranes labelled by 2 (these encoding all possible truth assignment of n variables of the input formula).

$$\begin{aligned}&[\,a_{i,2i-1}\,]_2 \longrightarrow [\,t_{i,i}\,]_2\,[\,f_{i,i}\,]_2, \text{ for } 1 \le i \le n\\ &[\,a_{i,j} \longrightarrow a_{i,j+1}\,]_2, \text{ for } 2 \le i \le n, 1 \le j \le 2i-2\\ &[\,[\,]_2\,[\,]_2\,]_1 \longrightarrow [\,[\,]_2\,]_1\;\;[\,[\,]_2\,]_1\\ &\left.\begin{matrix}[\,t_{i,j} \longrightarrow t_{i,j+1}\,]_2\\ [\,f_{i,j} \longrightarrow f_{i,j+1}\,]_2\end{matrix}\right\}, \text{ for } 1 \le i \le n, i \le j \le 2n-1\end{aligned}$$

5.3 Rules to produce exactly p copies of each truth assignment encoded by membranes labelled by 2.

$$\left.\begin{array}{l}[t_{i,2jn}\ F_i]_2 \longrightarrow\ t_{i,2jn+1}\ [\ ]_2\\ [f_{i,2jn}\ T_i]_2 \longrightarrow\ f_{i,2jn+1}\ [\ ]_2\\ t_{i,(2j+1)n}\ [\ \ ]_2 \longrightarrow\ [\ t_{i,(2j+1)n+1}\ ]_2\\ f_{i,(2j+1)n}\ [\ \ ]_2 \longrightarrow\ [\ f_{i,(2j+1)n+1}\ ]_2\end{array}\right\},\ \text{for } 1 \le i \le n,\ 1 \le j \le p-1$$

$$\left.\begin{array}{l}[t_{i,2np}\ F_i]_2 \longrightarrow\ \#\ [\ ]_2\\ [f_{i,2np}\ T_i]_2 \longrightarrow\ \#\ [\ ]_2\end{array}\right\},\ \text{for } 1 \le i \le n$$

$$\left.\begin{array}{l}[\ t_{i,(2j+1)n+k} \longrightarrow\ t_{i,(2j+1)n+k+1}\ ]_2\\ [\ f_{i,(2j+1)n+k} \longrightarrow\ f_{i,(2j+1)n+k+1}\ ]_2\\ [\ t_{i,2jn+k} \longrightarrow\ t_{i,2jn+k+1}\ ]_1\\ [\ f_{i,2jn+k} \longrightarrow\ f_{i,2jn+k+1}\ ]_1\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p-1,\\ 1 \le k \le n-1\end{array}$$

5.4 Rules to prepare the input formula for check clauses:

$$\left.\begin{array}{l}[\ x_{i,j,k} \longrightarrow x_{i,j,k+1}\ ]_2\\ [\ \overline{x}_{i,j,k} \longrightarrow \overline{x}_{i,j,k+1}\ ]_2\\ [\ x^*_{i,j,k} \longrightarrow x^*_{i,j,k+1}\ ]_2\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p,\\ 0 \le k \le 2np+n+n(j-1)+(i-1)-1\end{array}$$

5.5 Rules implementing the first checking stage.

$$\left.\begin{array}{l}[T_i\ x_{i,j,2np+n+n(j-1)+(i-1)}]_2 \longrightarrow c_{j,0}[\ \ ]_2\\ [T_i\ \overline{x}_{i,j,2np+n+n(j-1)+(i-1)}]_2 \longrightarrow \#[\ \ ]_2\\ [T_i\ x^*_{i,j,2np+n+n(j-1)+(i-1)}]_2 \longrightarrow \#[\ \ ]_2\\ [F_i\ x_{i,j,2np+n+n(j-1)+(i-1)}]_2 \longrightarrow \#[\ \ ]_2\\ [F_i\ \overline{x}_{i,j,2np+n+n(j-1)+(i-1)}]_2 \longrightarrow c_{j,0}[\ \ ]_2\\ [F_i\ x^*_{i,j,2np+n+n(j-1)+(i-1)}]_2 \longrightarrow \#[\ \ ]_2\end{array}\right\},\ \text{for } \begin{array}{l}1 \le i \le n,\\ 1 \le j \le p\end{array}$$

5.6 Rules implementing the second checking stage.

$$\begin{array}{l}[\ c_{j,k} \longrightarrow c_{j,k+1}\ ]_1,\ \text{for } 1 \le j \le p,\ 0 \le k \le np-2\\ c_{j,np-1}[\ \ ]_2 \longrightarrow\ [c_j]_2,\ \text{for } 1 \le j \le p\\ \left.\begin{array}{l}[\ \gamma_{4np+n}\ c_1]_2 \longrightarrow\ d_1[\ \ ]_2\\ [\ d_j\ c_{j+1}\ ]_2 \longrightarrow\ d_{j+1}[\ \ ]_2\\ d_j\ [\ \ ]_2 \longrightarrow\ [\ d_j\ ]_2\end{array}\right\},\ \text{for } 1 \le j \le p-1\end{array}$$

5.7 Rules to provide the correct answer of the system.

$$\begin{array}{l}[\ d_p\ ]_1 \longrightarrow\ d_p[\ \ ]_1\\ [\ \alpha_{4np+n+2p}\ d_p\ ]_0 \longrightarrow\ \texttt{yes}\ [\ \ ]_0\\ [\ \alpha_{4np+n+2p}\ \beta_{4np+n+2p+1}\ \ ]_0 \longrightarrow\ \texttt{no}\ [\ \ ]_0\end{array}$$

(6) the input membrane is the membrane labelled by 2 ($i_{in} = 2$) and the output region is the environment ($i_{out} = env$).

5 A formal verification

Let $\varphi = C_1 \wedge \ldots \wedge C_p$ an instance of SAT problem consisting of p clauses $C_j = l_{j,1} \vee \ldots \vee l_{j,r_j}$, $1 \le j \le p$, where $Var(\varphi) = \{x_1, \ldots, x_n\}$, and $l_{j,k} \in$

$\{x_i, \neg x_i | 1 \leq i \leq n\}$, $1 \leq j \leq p$, $1 \leq k \leq r_j$. Let us asume that the number of variables, n, and the number of clauses, p, of φ, are greater or equal to 2.

We consider the polynomial encoding (cod, s) from SAT in $\mathbf{\Pi}$ defined as follows: For each $\varphi \in I_{\mathtt{SAT}}$ with n variables and p clauses, $s(\varphi) = \langle n, p \rangle$ and

$$cod(\varphi) = \{x_{i,j,0} | x_i \in C_j\} \cup \{\overline{x}_{i,j,0} | \neg x_i \in C_j\} \cup \{x^*_{i,j,0} | x_i \notin C_j, \neg x_i \notin C_j\}$$

For instance, the formula $\varphi = (x_1 + x_2 + \overline{x}_3)(\overline{x}_2 + x_4)(\overline{x}_2 + x_3 + \overline{x}_4)$ is encoded as follows:

$$cod(\varphi) = \begin{pmatrix} x_{1,1,0} & x_{2,1,0} & \overline{x}_{3,1,0} & x^*_{4,1,0} \\ x^*_{1,2,0} & \overline{x}_{2,2,0} & x^*_{3,2,0} & x_{4,2,0} \\ x^*_{1,3,0} & \overline{x}_{2,3,0} & x_{3,3,0} & \overline{x}_{4,3,0} \end{pmatrix}$$

That is, j-th row ($1 \leq j \leq p$) represents the j-th clause C_j of φ. We denote $(cod(\varphi))^p_j$ the code of the clauses $C_j, \ldots, C_p$, that is, the expression containing from j-th row to p-th row. For instance,

$$cod(\varphi)^p_2 = \begin{pmatrix} x^*_{1,2,0} & \overline{x}_{2,2,0} & x^*_{3,2,0} & x_{4,2,0} \\ x^*_{1,3,0} & \overline{x}_{2,3,0} & x_{3,3,0} & \overline{x}_{4,3,0} \end{pmatrix}$$

We denote $(cod_k(\varphi))^p_j)$ the code $cod(\varphi)^p_j$ when the third index of the variables equal 3. For instance: row to p-th row. For instance,

$$cod_3(\varphi)^p_2 = \begin{pmatrix} x^*_{1,2,3} & \overline{x}_{2,2,3} & x^*_{3,2,3} & x_{4,2,3} \\ x^*_{1,3,3} & \overline{x}_{2,3,3} & x_{3,3,3} & \overline{x}_{4,3,3} \end{pmatrix}$$

We denote $(cod'_k(\varphi))^p_j)$ the code $cod(\varphi)^p_j$ when the third index of the variables equal 3. For instance: row to p-th row. For instance,

$$cod'_3(\varphi)^p_2 = \begin{pmatrix} {x^*}'_{1,2,3} & \overline{x}'_{2,2,3} & {x^*}'_{3,2,3} & x'_{4,2,3} \\ {x^*}'_{1,3,3} & \overline{x}'_{2,3,3} & x'_{3,3,3} & \overline{x}'_{4,3,3} \end{pmatrix}$$

We denote $(cod^*(\varphi))^p_j)$ the code $cod(\varphi)^p_j$ when the third index does not exist. For instance: row to p-th row. For instance,

$$cod^*(\varphi)^p_2 = \begin{pmatrix} {x^*}_{1,2} & \overline{x}_{2,2} & x^*_{3,2} & x_{4,2} \\ {x^*}_{1,3} & \overline{x}_{2,3} & x_{3,3} & \overline{x}_{4,3} \end{pmatrix}$$

The Boolean formula φ will be processed by the system $\Pi(s(\varphi)) + cod(\varphi)$. Next, we informally describe how that system works.

The solution proposed follows a brute force algorithm in the framework of recognizer P systems with active membranes, minimal cooperation in object evolution rules and division rules only for elementary membranes, and it consists of the following stages:

- *Generation stage*: using separation rules, beside other rules that make a "simulation" of division rules, we get all truth assignments for the variables $\{x_1, \ldots, x_n\}$ associated with φ are produced. Specifically, 2^n membranes labelled by 1 and 2^n labelled by 2 are generated. Each of the former ones encodes a truth assignment. This stage takes exactly $n + 2np$ steps, being n the number of variables of φ.

- *First Checking stage*: checking whether or not each clause of the input formula φ is satisfied by the truth assignments generated in the previous stage, encoded by each membrane labelled by 2. This stage takes exactly np steps, being n the number of the variables and p the number of clauses of φ.
- *Second Checking stage*: checking whether or not all clauses of the input formula φ are satisfied by some truth assignment encoded by a membrane labelled by 2. This stage takes exactly $np + 2p$ steps, being n the number of variables and p the number of clauses of φ.
- *Output stage*: the system sends to the environment the right answer according to the results of the previous stage. This stage takes 2 steps if the answer is `yes` and 3 steps if the answer is `no`.

5.1 Generation stage

Through this stage, all the different truth assignments for the variables associated with the Boolean formula φ will be generated within membranes labelled by 2, by the applications of rules from **5.2** and **5.3**. In the first $2n$ steps, 2^n membranes labelled by 1 and 2^n membranes labelled by 2, alternating between the division of membranes labelled by 2 (in odd steps) and the division of membranes labelled by 1 (in even steps).

Proposition 1. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a_0) *For each $2k$ ($0 \leq k \leq n-1$) at configuration $\mathcal{C}_{2k}$ we have the following:*
- $\mathcal{C}_{2k}(0) = \{\alpha_{2k}, \beta_{2k}\}$
- *There are 2^k empty membranes labelled by 1.*
- *There are 2^k membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset $cod_{2k}(\varphi)$;*
  - ⋆ *an object γ_{2k}; and*
  - ⋆ *p copies of every T_i and F_i, $1 \leq i \leq n$.*
  - ⋆ *objects $a_{i,2k+1}$, $k+1 \leq i \leq n$; and*
  - ⋆ *a different subset $\{r_{1,j}, \ldots, r_{k,j}\}$, $k+1 \leq j \leq 2k$, being $r \in \{t, f\}$.*

(a_1) *For each $2k+1$ ($0 \leq j \leq n-1$) at configuration $\mathcal{C}_{2k+1}$ we have the following:*
- $\mathcal{C}_{2k+1}(0) = \{\alpha_{2k+1}, \beta_{2k+1}\}$
- *There are 2^k empty membranes labelled by 1.*
- *There are 2^{k+1} membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset $cod_{2k+1}(\varphi)$;*
  - ⋆ *an object γ_{2k+1}; and*
  - ⋆ *p copies of every T_i and F_i, $1 \leq i \leq n$.*
  - ⋆ *objects $a_{i,2(k+1)}$, $k+1 \leq i \leq n$; and*
  - ⋆ *a different subset $\{r_{1,j}, \ldots, r_{k+1,j}\}$, $k+1 \leq j \leq 2k+1$, being $r \in \{t, f\}$.*

(b) *$\mathcal{C}_{2n}(0) = \{\alpha_{2n}, \beta_{2n}\}$, and in $\mathcal{C}_{2n}$ there are 2^n empty membranes labelled by 1; and 2^n membranes labelled by 2, such that each of them contains the input multiset $cod_{2n}(\varphi)$, p copies of every T_i and F_i ($1 \leq i \leq n$), an object γ_{2n} and a different subset of objects $r_{i,2n+1-i}$, $1 \leq i \leq n$.*

Proof. (a) is going to be demonstrated by induction on k

- The base case $k = 0$ is trivial because:
  (a_0) at the initial configuration $\mathcal{C}_0$ we have: $\mathcal{C}_0(0) = \{\alpha_0, \beta_0\}$ and there exists a single empty membrane labelled by 1 containing ; and a single membrane labelled by 2 containing the input multiset $cod(\varphi)$, an object γ_0, p copies of T_i and F_i, being $1 \leq i \leq n$, the objects $a_{1,1}, \ldots, a_{n,1}$. Then, configuration $\mathcal{C}_0$ yields configuration $\mathcal{C}_1$ by applying the rules:
  $[\ a_{1,1}\ ]_2 \to [\ t_{1,1}\ ]_2\ [\ f_{1,1}\ ]_2$
  $[\ a_{i,1} \to a_{i,2}\ ]_2$, for $k+1 \leq i \leq n$
  $[\ \alpha_0 \to \alpha_1\ ]_0$
  $[\ \beta_0 \to \beta_1\ ]_0$
  $[\ \gamma_0 \to \gamma_1\ ]_2$
  $\left.\begin{array}{l} [\ x_{i,j,0} \to x_{i,j,1}\ ]_2 \\ [\ \overline{x}_{i,j,0} \to \overline{x}_{i,j,1}\ ]_2 \\ [\ x^*_{i,j,0} \to x^*_{i,j,1}\ ]_2 \end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$
  (a_1) at $\mathcal{C}_1$ we have $\mathcal{C}_1(0) = \{\alpha_1, \beta_1\}$ and there exists a single empty membrane labelled by 1; and two membranes labelled by 2 containing the input multiset $cod_1(\varphi)$, an object γ_1, p copies of T_i and F_i, being $1 \leq i \leq n$, the objects $a_{2,2}, \ldots, a_{n,2}$ and one with the object $t_{1,1}$ and the other one with the object $f_{1,1}$. Then, the configuration $\mathcal{C}_1$ yields configuration $\mathcal{C}_2$ by applying the rules:
  $[\ t_{1,1} \to t_{1,2}\ ]_2$
  $[\ f_{1,1} \to f_{1,2}\ ]_2$
  $[\ [\ \ ]_2\ [\ \ ]_2]_1 \to [\ [\ \ ]_2]_1\ [\ [\ \ ]_2]_1$
  $[a_{i,2} \to a_{i,3}\ ]_2$, for $2 \leq i \leq n$
  $[\ \alpha_1 \to \alpha_2\ ]_0$
  $[\ \beta_1 \to \beta_2\ ]_0$
  $[\ \gamma_1 \to \gamma_2\ ]_2$
  $\left.\begin{array}{l} [\ x_{i,j,1} \to x_{i,j,2}\ ]_2 \\ [\ \overline{x}_{i,j,1} \to \overline{x}_{i,j,2}\ ]_2 \\ [\ x^*_{i,j,1} \to x^*_{i,j,2}\ ]_2 \end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$
  Thus, $\mathcal{C}_2(0) = \{\alpha_2, \beta_2\}$, and there exist two empty membranes labelled by 1; and two membranes labelled by 2 containing the input multiset $cod_2(\varphi)$, an object γ_2, p copies of T_i and F_i, being $1 \leq i \leq n$, the objects $a_{2,3}, \ldots, a_{n,3}$ and one with the object $t_{1,2}$ and the other one with the object $f_{1,2}$. Hence, the result holds for $k = 1$.
- Supposing, by induction, result is true for k $(0 \leq k \leq n-1)$
  - $\mathcal{C}_{2k}(0) = \{\alpha_{2k}, \beta_{2k}\}$
  - In $\mathcal{C}_{2k}$ there are 2^k empty membranes labelled by 1.
  - In $\mathcal{C}_{2k}$ there are 2^k membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2k}(\varphi)$;
    - ⋆ an object γ_{2k};
    - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
    - ⋆ objects $a_{i,2k+1}$, $k+1 \leq i \leq n$; and

⋆ a different subset $\{r_{1,j},\ldots,r_{k,j}\}$, $k+1 \leq j \leq 2k$, being $r \in \{t,f\}$.

Then, configuration $\mathcal{C}_{2k}$ yields configuration $\mathcal{C}_{2k+1}$ by applying the rules:

$[\ a_{k,2k+1}\ ]_2 \rightarrow [\ t_{k,k}\ ]_2\ [\ f_{k,k}\ ]_2$

$[\ a_{i,2k+1} \rightarrow a_{i,2k+2}\ ]_2$,for $k+1 \leq i \leq n$

$\left.\begin{array}{l}[\ t_{i,j} \rightarrow t_{i,j+1}\ ]_2\\ [\ f_{i,j} \rightarrow f_{i,j+1}\ ]_2\end{array}\right\}$ for $1 \leq i \leq k-1, k+1 \leq j \leq 2k$

$[\ \alpha_{2k} \rightarrow \alpha_{2k+1}\ ]_0$

$[\ \beta_{2k} \rightarrow \beta_{2k+1}\ ]_0$

$[\ \gamma_{2k} \rightarrow \gamma_{2k+1}\ ]_2$

$\left.\begin{array}{l}[\ x_{i,j,2k} \rightarrow x_{i,j,2k+1}\ ]_2\\ [\ \overline{x}_{i,j,1} \rightarrow \overline{x}_{i,j,2k+1}\ ]_2\\ [\ x^*_{i,j,1} \rightarrow x^*_{i,j,2k+1}\ ]_2\end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

Therefore, the following holds

- $\mathcal{C}_{2k+1}(0) = \{\alpha_{2k+1}, \beta_{2k+1}\}$
- In $\mathcal{C}_{2k+1}$ there are 2^k empty membranes labelled by 1.
- In $\mathcal{C}_{2k+1}$ there are 2^{k+1} membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod_{2k+1}(\varphi)$;
  - ⋆ an object γ_{2k+1};
  - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
  - ⋆ objects $a_{i,2(k+1)}$, $k+1 \leq i \leq n$; and
  - ⋆ a different subset $\{r_{1,j},\ldots,r_{k+1,j}\}$, $k+1 \leq j \leq 2k+1$, being $r \in \{t,f\}$.

Then, configuration $\mathcal{C}_{2k+1}$ yields configuration $\mathcal{C}_{2(k+1)}$ by applying the rules:

$\left.\begin{array}{l}[\ t_{i,j} \rightarrow t_{i,j+1}\ ]_2\\ [\ f_{i,j} \rightarrow f_{i,j+1}\ ]_2\end{array}\right\}$ for $1 \leq i \leq k+1, k+1 \leq j \leq 2k+1$

$[\ [\ \ ]_2\ [\ \ ]_2]_1 \rightarrow [\ [\ \ ]_2]_1\ [\ [\ \ ]_2]_1$

$[\ a_{i,2(k+1)} \rightarrow a_{i,2(k+1)+1}\ ]_2$,for $k+1 \leq i \leq n$

$[\ \alpha_{2k+1} \rightarrow \alpha_{2(k+1)}\ ]_0$

$[\ \beta_{2k+1} \rightarrow \beta_{2(k+1)}\ ]_0$

$[\ \gamma_{2k+1} \rightarrow \gamma_{2(k+1)}\ ]_2$

$\left.\begin{array}{l}[\ x_{i,j,2k+1} \rightarrow x_{i,j,2k+2}\ ]_2\\ [\ \overline{x}_{i,j,2k+1} \rightarrow \overline{x}_{i,j,2k+2}\ ]_2\\ [\ x^*_{i,j,2k+1} \rightarrow x^*_{i,j,2k+2}\ ]_2\end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

Therefore, the following holds

- $\mathcal{C}_{2(k+1)}(0) = \{\alpha_{2(k+1)}, \beta_{2(k+1)}\}$
- In $\mathcal{C}_{2(k+1)}$ there are 2^{k+1} empty membranes labelled by 1.
- In $\mathcal{C}_{2(k+1)}$ there are 2^{k+1} membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod_{2(k+1)}(\varphi)$;
  - ⋆ an object $\gamma_{2(k+1)}$;
  - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$.
  - ⋆ objects $a_{i,2(k+1)+1}$, $k+1 \leq i \leq n$; and
  - ⋆ a different subset $\{r_{1,j},\ldots,r_{k+1,j}\}$, $k+1 \leq j \leq 2(k+1)+1$.

Hence, the result holds for $k+1$.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{2n-1}$ holds:
  - $\mathcal{C}_{2n-1}(0) = \{\alpha_{2n-1}, \beta_{2n-1}\}$
  - In $\mathcal{C}_{2n-1}$ there are 2^{n-1} empty membranes labelled by 1.
  - In $\mathcal{C}_{2n-1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2n-1}p(\varphi)$;
    - ⋆ an object γ_{2n-1};
    - ⋆ p copies of T_i and F_i, $1 \leq i \leq n$; and
    - ⋆ a different subset of objects $r_{i,2n-i}$, $1 \leq i \leq n$.

    Then, configuration $\mathcal{C}_{2n-1}$ yields $\mathcal{C}_{2n}$ by applying the rules:

$$\left.\begin{array}{l}[\ t_{i,2n-i} \to t_{i,2n+1-i}\ ]_2 \\ [\ f_{i,2n-i} \to f_{i,2n+1-1}\ ]_2\end{array}\right\} \text{ for } 1 \leq i \leq n$$

$$[\ [\quad]_2\,[\quad]_2]_1 \to [\ [\quad]_2]_1\ [\ [\quad]_2]_1$$

$$[\ \alpha_{2n-1} \to \alpha_{2n}\ ]_0$$

$$[\ \beta_{2n-1} \to \beta_{2n}\ ]_0$$

$$[\ \gamma_{2n-1} \to \gamma_{2n}\ ]_2$$

$$\left.\begin{array}{l}[\ x_{i,j,2n-1} \to x_{i,j,2n}\ ]_2 \\ [\ \overline{x}_{i,j,2n-1} \to \overline{x}_{i,j,2n}\ ]_2 \\ [\ x^*_{i,j,2n-1} \to x^*_{i,j,2n}\ ]_2\end{array}\right\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p$$

    Then, we have $\mathcal{C}_{2n}(0) = \{\alpha_{2n}, \beta_{2n}\}$, and there exist 2^n empty membranes labelled by 1; and 2^n membranes labelled by 2 containing the input multiset $cod_{2n}(\varphi)$, an object γ_{2n}, p copies of T_i and F_i, being $1 \leq i \leq n$ and a different multiset of objects $r_{i,2n+1-i}$, being $1 \leq i \leq n$.

□

When the tree structure is created, we start assigning a truth assignment to each branch. It is executed in the next $2np-n$ steps. The last n steps are different from the previous ones, so they deserve another proposition of the following one.

Proposition 2. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a_0) *For each k $(1 \leq k \leq n)$ and l $(0 \leq l \leq p-1)$ at configuration $\mathcal{C}_{2n+2ln+k}$ we have the following:*
- *$\mathcal{C}_{2n+2ln+k}(0) = \{\alpha_{2n+2ln+k}, \beta_{2n+2ln+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains a different subset of objects $r_{i,2n+2ln+k-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.*
- *There are 2^n membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset $cod_{2n+2ln+k}(\varphi)$;*
  - ⋆ *an object $\gamma_{2n+2ln+k}$;*
  - ⋆ *p copies of every T_i and F_i, $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding t_i or f_i object; otherwise, there are $p-l$ copies if $k+1 \leq i \leq n$, $p-l-1$ otherwise; and*

  - ⋆ *objects* $r_{i,2n+2ln+k-i+1}$, $k+1 \leq i \leq n$, *being* $r \in \{t, f\}$.

(a_1) *For each* k $(1 \leq k \leq n)$ *and* l $(0 \leq l \leq p-1)$ *at configuration* $\mathcal{C}_{3n+2ln+k}$ *we have the following:*

- $\mathcal{C}_{2n+2ln+k}(0) = \{\alpha_{2n+2ln+k}, \beta_{2n+2ln+k}\}$
- *There are* 2^n *membranes labelled by 1 such that each of them contains a different subset of objects* $r_{i,3n+ln+k-i+1}$, $k+1 \leq i \leq n$, *being* $r \in \{t, f\}$.
- *There are* 2^n *membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset* $cod_{3n+2ln+k}(\varphi)$*;*
  - ⋆ *an object* $\gamma_{3n+2ln+k}$*;*
  - ⋆ p *copies of every* T_i *and* F_i, $1 \leq i \leq n$ *if the truth assignment associated to the branch contains its corresponding* t_i *or* f_i *object, and* $p-l$ *copies otherwise; and*
  - ⋆ *objects* $r_{i,3n+2ln+k-i+1}$, $1 \leq i \leq k$, *being* $r \in \{t, f\}$.

(b) $\mathcal{C}_{2np}(0) = \{\alpha_{2np}, \beta_{2np}\}$, *and in* $\mathcal{C}_{2np}$ *there are* 2^n *empty membranes labelled by 1; and* 2^n *membranes labelled by 2 such that each of them contains the input multiset* $cod_{2np}(\varphi)$*, an object* γ_{2np}*,* p *copies of every* T_i *and* F_i, $1 \leq i \leq n$ *if the truth assignment associated to the branch contains its corresponding* t_i *or* f_i *object, and 1 object otherwise and objects* $r_{i,2np-i+1}$, $1 \leq i \leq n$, *being* $r \in \{t, f\}$*, that is, the truth assignment associated with the branch.*

Proof. (a) is going to be demonstrated by induction on l

- The base case $l = 0$ is going to be demonstrated by induction on k

  (a_0) The base case $k = 1$ is trivial because:

  - at configuration $\mathcal{C}_{2n}$ we have: $\mathcal{C}_{2n}(0) = \{\alpha_{2n}, \beta_{2n}\}$ and there exist 2^n empty membranes labelled by 1 containing ; and 2^n membranes labelled by 2 containing the input multiset $cod_{2n}(\varphi)$, an object γ_{2n}, p copies of T_i and F_i, being $1 \leq i \leq n$ and a different subset of objects $r_{i,2n-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, the corresponding truth assignment of the branch. Then, configuration $\mathcal{C}_{2n}$ yields configuration $\mathcal{C}_{2n+1}$ by applying the rules:

$$
\begin{array}{l}
[\ t_{i,2n}\ F_i\ ]_2 \to t_{i,2n+1}[\ \ ]_2 \\
[\ f_{i,2n}\ T_i\ ]_2 \to f_{i,2n+1}[\ \ ]_2 \\
\left.\begin{array}{l} [\ t_{i,2n+1-i} \to t_{i,2n+2-i}\ ]_2 \\ [\ f_{i,2n+1-i} \to f_{i,2n+2-1}\ ]_2 \end{array}\right\} \text{for } 2 \leq i \leq n \\
[\ \alpha_{2n} \to \alpha_{2n+1}\ ]_0 \\
[\ \beta_{2n} \to \beta_{2n+1}\ ]_0 \\
[\ \gamma_{2n} \to \gamma_{2n+1}\ ]_2 \\
\left.\begin{array}{l} [\ x_{i,j,2n} \to x_{i,j,2n+1}\ ]_2 \\ [\ \overline{x}_{i,j,2n} \to \overline{x}_{i,j,2n+1}\ ]_2 \\ [\ x^*_{i,j,2n} \to x^*_{i,j,2n+1}\ ]_2 \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

    Thus, $\mathcal{C}_{2n+1}(0) = \{\alpha_{2n+1}, \beta_{2n+1}\}$, and there exist 2^n membranes labelled by 1 containing an object $r_{1,2n+1}$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing the input multiset $cod_{2n+1}(\varphi)$, an object γ_{2n+1}, p copies of T_i and F_i, being $2 \leq i \leq n$, and $p-1$ copies of T_1

(respectively, F_i) if object $f_{1,2n}$ (resp., $t_{1,2n}$) was within membrane labelled by 2 at configuration $\mathcal{C}_{2n}$, and p copies of F_1 (resp., T_1), and a different subset of objects $r_{i,2n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k $(1 \leq k \leq n)$
  - $\mathcal{C}_{2n+k}(0) = \{\alpha_{2n+k}, \beta_{2n+k}\}$
  - In $\mathcal{C}_{2n+k}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,2n+k-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2n+k}(\varphi)$;
    - ⋆ an object γ_{2n+k};
    - ⋆ p copies of every T_i and F_i, $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding t_i or f_i object, $p-1$ objects T_i and F_i $(1 \leq i \leq k)$ otherwise; and
    - ⋆ a different subset of objects $r_{i,2n+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{2n+k}$ yields configuration $\mathcal{C}_{2n+k+1}$ by applying the rules:

$$
\begin{array}{l}
[\ t_{k+1,2n}\ F_{k+1}\ ]_2 \to t_{k+1,2n+1}[\ \ ]_2 \\
[\ f_{k+1,2n}\ T_{k+1}\ ]_2 \to f_{k+1,2n+1}[\ \ ]_2 \\
\left.\begin{array}{l} [\ t_{i,2n+k-i+1} \to t_{i,2n+k-i+2}\ ]_2 \\ [\ f_{i,2n+k-i+1} \to f_{i,2n+k-i+2}\ ]_2 \end{array}\right\} \text{for } k+2 \leq i \leq n \\
\left.\begin{array}{l} [\ t_{i,2n+k-i+1} \to t_{i,2n+k-i+2}\ ]_1 \\ [\ f_{i,2n+k-i+1} \to f_{i,2n+k-i+2}\ ]_1 \end{array}\right\} \text{for } 1 \leq i \leq k \\
[\ \alpha_{2n+k} \to \alpha_{2n+k+1}\ ]_0 \\
[\ \beta_{2n+k} \to \beta_{2n+k+1}\ ]_0 \\
[\ \gamma_{2n+k} \to \gamma_{2n+k+1}\ ]_2 \\
\left.\begin{array}{l} [\ x_{i,j,2n+k} \to x_{i,j,2n+k+1}\ ]_2 \\ [\ \overline{x}_{i,j,2n+k} \to \overline{x}_{i,j,2n+k+1}\ ]_2 \\ [\ x^*_{i,j,2n+k} \to x^*_{i,j,2n+k+1}\ ]_2 \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

    Therefore, the following holds
  - $\mathcal{C}_{2n+k+1}(0) = \{\alpha_{2n+k+1}, \beta_{2n+k+1}\}$
  - In $\mathcal{C}_{2n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,2n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2n+k+1}(\varphi)$;
    - ⋆ an object γ_{2n+k+1};
    - ⋆ p copies of every T_i and F_i, $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding t_i or f_i object, $p-1$ objects T_i and F_i $(1 \leq i \leq k+1)$ otherwise; and
    - ⋆ a different subset of objects $r_{i,2n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.

(a_1) The base case $k = 1$ is trivial because:

- at configuration $\mathcal{C}_{3n}$ we have $\mathcal{C}_{3n}(0) = \{\alpha_{3n}, \beta_{3n}\}$ and there exist 2^n membranes labelled by 1 containing and a different subset of objects $r_{i,3n+1-i}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the corresponding truth assignment of the branch; and 2^n membranes labelled by 2 containing the input multiset $cod_{3n}(\varphi)$, an object γ_{3n}, p copies of T_i and F_i, being $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding object t_i or f_i, $p-1$ objects otherwise. Then, configuration $\mathcal{C}_{3n}$ yields configuration $\mathcal{C}_{3n+1}$ by applying the rules:

$$
\begin{array}{l}
t_{1,3n}\ [\ \ ]_2 \to [\ t_{1,3n+1}\ ]_2 \\
f_{1,3n}\ [\ \ ]_2 \to [\ f_{1,3n+1}\ ]_2 \\
\left.\begin{array}{l} [\ t_{i,3n-i+1} \to t_{i,3n-i+2}\ ]_1 \\ [\ f_{i,3n-i+1} \to f_{i,3n-i+2}\ ]_1 \end{array}\right\} \text{ for } 2 \leq i \leq n \\
[\ \alpha_{3n} \to \alpha_{3n+1}\ ]_0 \\
[\ \beta_{3n} \to \beta_{3n+1}\ ]_0 \\
[\ \gamma_{3n} \to \gamma_{3n+1}\ ]_2 \\
\left.\begin{array}{l} [\ x_{i,j,3n} \to x_{i,j,3n+1}\ ]_2 \\ [\ \overline{x}_{i,j,3n} \to \overline{x}_{i,j,3n+1}\ ]_2 \\ [\ x^*_{i,j,3n} \to x^*_{i,j,3n+1}\ ]_2 \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

  Thus, $\mathcal{C}_{3n+1}(0) = \{\alpha_{3n+1}, \beta_{3n+1}\}$, and there exist 2^n membranes labelled by 1 containing a different subset of objects $r_{i,3n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing the input multiset $cod_{3n+1}(\varphi)$, an object γ_{3n+1}, p copies of T_i and F_i, being $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding object t_i or f_i, $p-1$ objects otherwise and an object $r_{1,3n+1}$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k $(1 \leq k \leq n)$
  - $\mathcal{C}_{3n+k}(0) = \{\alpha_{3n+k}, \beta_{3n+k}\}$
  - In $\mathcal{C}_{3n+k}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,3n+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{3n+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{3n+k}(\varphi)$;
    - ⋆ an object γ_{3n+k};
    - ⋆ p copies of every T_i and F_i for $1 \leq i \leq n$ or their corresponding t_i or f_i is assigned to that branch, $p - l$ copies otherwise; and
    - ⋆ a different subset of objects $r_{i,3n+k-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{3n+k}$ yields configuration $\mathcal{C}_{3n+k+1}$ by applying the rules:

$$
\begin{array}{l}
t_{k+1,3n}\ [\ \ ]_2 \to [\ t_{k+1,3n+1}\ ]_2 \\
f_{k+1,3n}\ [\ \ ]_2 \to [\ f_{k+1,3n+1}\ ]_2 \\
\left.\begin{array}{l} [\ t_{i,3n+k-i+1} \to t_{i,3n+k-i+2}\ ]_1 \\ [\ f_{i,3n+k-i+1} \to f_{i,3n+k-i+2}\ ]_1 \end{array}\right\} \text{ for } k+2 \leq i \leq n \\
\left.\begin{array}{l} [\ t_{i,3n+k-i+1} \to t_{i,3n+k-i+2}\ ]_2 \\ [\ f_{i,3n+k-i+1} \to f_{i,3n+k-i+2}\ ]_2 \end{array}\right\} \text{ for } 1 \leq i \leq k
\end{array}
$$

$$\left.\begin{array}{l}[\ \alpha_{3n+k} \to \alpha_{3n+k+1}\ ]_0\\ [\ \beta_{3n+k} \to \beta_{3n+k+1}\ ]_0\\ [\ \gamma_{3n+k} \to \gamma_{3n+k+1}\ ]_2\\ [\ x_{i,j,3n+k} \to x_{i,j,3n+k+1}\ ]_2\\ [\ \overline{x}_{i,j,3n+k} \to \overline{x}_{i,j,3n+k+1}\ ]_2\\ [\ x^*_{i,j,3n+k} \to x^*_{i,j,3n+k+1}\ ]_2\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p$$

Therefore, the following holds

- $\mathcal{C}_{3n+k+1}(0) = \{\alpha_{3n+k+1}, \beta_{3n+k+1}\}$
- In $\mathcal{C}_{3n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,3n+k-i+2}$, $k+2 \le i \le n$, being $r \in \{t, f\}$.
- In $\mathcal{C}_{3n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod_{3n+k+1}(\varphi)$;
  - ⋆ an object γ_{3n+k+1};
  - ⋆ p copies of every T_i and F_i for $1 \le i \le n$ or the corresponding t_i or f_i is assigned to that branch, $p-l$ copies otherwise; and
  - ⋆ a different subset of objects $r_{i,3n+k-i+2}$, $1 \le i \le k+1$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for l $(0 \le l \le p-1)$

  (a_0) The base case $k = 1$ is trivial because:

  - at configuration $\mathcal{C}_{2n+(l+1)n}$[1] we have: $\mathcal{C}_{2n+(l+1)n}(0) = \{\alpha_{2n+(l+1)n}, \beta_{2n+(l+1)n}\}$ and there exist 2^n empty membranes labelled by 1; and 2^n membranes labelled by 2 containing the input multiset $cod_{2n+(l+1)n}(\varphi)$, an object $\gamma_{2n+(l+1)n}$, p copies of T_i and F_i, being $1 \le i \le n$, and $p-l$ copies for T_i (resp. F_i) objects that are in a branch with an object f_i (resp. t_i) and a different subset of objects $r_{i,2n+(l+1)n-i+1}$, $1 \le i \le n$, being $r \in \{t, f\}$, the corresponding truth assignment of the branch. Then, configuration $\mathcal{C}_{2n+(l+1)n}$ yields configuration $\mathcal{C}_{2n+(l+1)n+1}$ by applying the rules:

$$\left.\begin{array}{l}[\ t_{i,2n+(l+1)n}\ F_i\ ]_2 \to t_{i,2n+(l+1)n+1}[\ \ ]_2\\ [\ f_{i,2n+(l+1)n}\ T_i\ ]_2 \to f_{i,2n+(l+1)n+1}[\ \ ]_2\\ [\ t_{i,2n+(l+1)n+1-i} \to t_{i,2n+(l+1)n+2-i}\ ]_2\\ [\ f_{i,2n+(l+1)n+1-i} \to f_{i,2n+(l+1)n+2-i}\ ]_2\end{array}\right\} \text{for } 2 \le i \le n$$

$$\left.\begin{array}{l}[\ \alpha_{2n+(l+1)n} \to \alpha_{2n+(l+1)n+1}\ ]_0\\ [\ \beta_{2n+(l+1)n} \to \beta_{2n+(l+1)n+1}\ ]_0\\ [\ \gamma_{2n+(l+1)n} \to \gamma_{2n+(l+1)n+1}\ ]_2\\ [\ x_{i,j,2n+(l+1)n} \to x_{i,j,2n+(l+1)n+1}\ ]_2\\ [\ \overline{x}_{i,j,2n+(l+1)n} \to \overline{x}_{i,j,2n+(l+1)n+1}\ ]_2\\ [\ x^*_{i,j,2n+(l+1)n} \to x^*_{i,j,2n+(l+1)n+1}\ ]_2\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p$$

  Thus, $\mathcal{C}_{2n+(l+1)n+1}(0) = \{\alpha_{2n+(l+1)n+1}, \beta_{2n+(l+1)n+1}\}$, and there exist 2^n membranes labelled by 1 containing and an object $r_{1,2n+(l+1)n+1}$,

[1] Note that $(l+1)n = ln + n$, and it has been demonstrated in the first step of the induction that it is correct.

being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing the input multiset $cod_{2n+(l+1)n+1}(\varphi)$, an object $\gamma_{2n+(l+1)n+1}$, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, otherwise $p-l$ copies of F_i (resp. T_i) if $2 \leq i \leq n$, $p-l-1$ otherwise and a different subset of objects $r_{i,2n+(l+1)n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k $(1 \leq k \leq n)$
  - $\mathcal{C}_{2n+(l+1)n+k}(0) = \{\alpha_{2n+(l+1)n+k}, \beta_{2n+(l+1)n+k}\}$
  - In $\mathcal{C}_{2n+(l+1)n+k}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,2n+(l+1)n+k-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+(l+1)n+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2n+(l+1)n+k}(\varphi)$;
    - ⋆ an object $\gamma_{2n+(l+1)n+k}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, otherwise $p-l$ copies of F_i (resp. T_i) if $k+1 \leq i \leq n$, $p-l-1$ otherwise; and
    - ⋆ a different subset of objects $r_{i,2n+(l+1)n+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{2n+k}$ yields configuration $\mathcal{C}_{2n+(l+1)n+k+1}$ by applying the rules:

$$
\begin{array}{l}
[\ t_{k+1,2n+(l+1)n}\ F_{k+1}\ ]_2 \to t_{k+1,2n+(l+1)n+1}[\ \ ]_2 \\
[\ f_{k+1,2n+(l+1)n}\ T_{k+1}\ ]_2 \to f_{k+1,2n+(l+1)n+1}[\ \ ]_2 \\
\left.\begin{array}{l} [\ t_{i,2n+(l+1)n+k-i+1} \to t_{i,2n+k-i+2}\ ]_2 \\ [\ f_{i,2n+(l+1)n+k-i+1} \to f_{i,2n+k-i+2}\ ]_2 \end{array}\right\} \text{ for } k+2 \leq i \leq n \\
\left.\begin{array}{l} [\ t_{i,2n+(l+1)n+k-i+1} \to t_{i,2n+(l+1)n+k-i+2}\ ]_1 \\ [\ f_{i,2n+(l+1)n+k-i+1} \to f_{i,2n+(l+1)n+k-i+2}\ ]_1 \end{array}\right\} \text{ for } 1 \leq i \leq k \\
[\ \alpha_{2n+(l+1)n+k} \to \alpha_{2n+(l+1)n+k+1}\ ]_0 \\
[\ \beta_{2n+(l+1)n+k} \to \beta_{2n+(l+1)n+k+1}\ ]_0 \\
[\ \gamma_{2n+(l+1)n+k} \to \gamma_{2n+(l+1)n+k+1}\ ]_2 \\
\left.\begin{array}{l} [\ x_{i,j,2n+(l+1)n+k} \to x_{i,j,2n+(l+1)n+k+1}\ ]_2 \\ [\ \overline{x}_{i,j,2n+(l+1)n+k} \to \overline{x}_{i,j,2n+(l+1)n+k+1}\ ]_2 \\ [\ x^*_{i,j,2n+(l+1)n+k} \to x^*_{i,j,2n+(l+1)n+k+1}\ ]_2 \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

    Therefore, the following holds
  - $\mathcal{C}_{2n+(l+1)n+k+1}(0) = \{\alpha_{2n+(l+1)n+k+1}, \beta_{2n+(l+1)n+k+1}\}$
  - In $\mathcal{C}_{2n+(l+1)n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,2n+(l+1)n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{2n+(l+1)n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2n+(l+1)n+k+1}(\varphi)$;
    - ⋆ an object $\gamma_{2n+(l+1)n+k+1}$;

  - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, otherwise $p-l$ copies of F_i (resp. T_i) if $k+2 \leq i \leq n$, $p-l-1$ otherwise; and
  - ⋆ a different subset of objects $r_{i,2n+(l+1)n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.

(a_1) The base case $k = 1$ is trivial because:

- at configuration $\mathcal{C}_{3n+(l+1)n}$ we have $\mathcal{C}_{3n+(l+1)n}(0) = \{\alpha_{3n+(l+1)n}, \beta_{3n+(l+1)n}\}$ and there exist 2^n membranes labelled by 1 containing a different subset of objects $r_{i,3n+(l+1)n-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the corresponding truth assignment of the branch; and 2^n membranes labelled by 2 containing the input multiset $cod_{3n+(l+1)n}(\varphi)$, an object $\gamma_{3n+(l+1)n}$ and p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i). Then, configuration $\mathcal{C}_{3n+(l+1)n}$ yields configuration $\mathcal{C}_{3n+(l+1)n+1}$ by applying the rules:

$$
\begin{array}{l}
t_{1,3n+(l+1)n}\ [\ \ ]_2 \to [\ t_{1,3n+(l+1)n+1}\ ]_2 \\
f_{1,3n+(l+1)n}\ [\ \ ]_2 \to [\ f_{1,3n+(l+1)n+1}\ ]_2 \\
\left.\begin{array}{l}
[\ t_{i,3n+(l+1)n-i+1} \to t_{i,3n+(l+1)n-i+2}\ ]_1 \\
[\ f_{i,3n+(l+1)n-i+1} \to f_{i,3n+(l+1)n-i+2}\ ]_1
\end{array}\right\} \text{ for } 2 \leq i \leq n \\
[\ \alpha_{3n+(l+1)n} \to \alpha_{3n+(l+1)n+1}\ ]_0 \\
[\ \beta_{3n+(l+1)n} \to \beta_{3n+(l+1)n+1}\ ]_0 \\
[\ \gamma_{3n+(l+1)n} \to \gamma_{3n+(l+1)n+1}\ ]_2 \\
\left.\begin{array}{l}
[\ x_{i,j,3n+(l+1)n} \to x_{i,j,3n+(l+1)n+1}\ ]_2 \\
[\ \overline{x}_{i,j,3n+(l+1)n} \to \overline{x}_{i,j,3n+(l+1)n+1}\ ]_2 \\
[\ x^*_{i,j,3n+(l+1)n} \to x^*_{i,j,3n+(l+1)n+1}\ ]_2
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

  Thus, $\mathcal{C}_{3n+(l+1)n+1}(0) = \{\alpha_{3n+(l+1)n+1}, \beta_{3n+(l+1)n+1}\}$, and there exist 2^n membranes labelled by 1 containing a different subset of objects $r_{i,3n+(l+1)n-i+2}$, $2 \leq i \leq n$, being $r \in \{t, f\}$; and 2^n membranes labelled by 2 containing the input multiset $cod_{3n+(l+1)n+1}(\varphi)$, an object $\gamma_{3n+(l+1)n+1}$, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i) and an object $r_{1,3n+(l+1)n+1}$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k ($1 \leq k \leq n$)
  - $\mathcal{C}_{3n+(l+1)n+k}(0) = \{\alpha_{3n+(l+1)n+k}, \beta_{3n+(l+1)n+k}\}$
  - In $\mathcal{C}_{3n+(l+1)n+k}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,3n+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{3n+(l+1)n+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{3n+(l+1)n+k}(\varphi)$;
    - ⋆ an object $\gamma_{3n+(l+1)n+k}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p-l$ copies of F_i (resp. T_i)

  - ⋆ a different subset of objects $r_{i,3n+(l+1)n-i+1}$, $1 \leq i \leq k$, being $r \in \{t, f\}$.

  Then, configuration $\mathcal{C}_{3n+(l+1)n+k}$ yields configuration $\mathcal{C}_{3n+(l+1)n+k+1}$ by applying the rules:

$$t_{k+1,3n+(l+1)n}\ [\ \ ]_2 \to [\ t_{k+1,3n+(l+1)n+1}\ ]_2$$
$$f_{k+1,3n+(l+1)n}\ [\ \ ]_2 \to [\ f_{k+1,3n+(l+1)n+1}\ ]_2$$
$$\left.\begin{array}{l}[\ t_{i,3n+(l+1)n+k-i+1} \to t_{i,3n+(l+1)n+k-i+2}\ ]_1 \\ [\ f_{i,3n+(l+1)n+k-i+1} \to f_{i,3n+(l+1)n+k-i+2}\ ]_1\end{array}\right\} \text{ for } k+2 \leq i \leq n$$
$$\left.\begin{array}{l}[\ t_{i,3n+(l+1)n+k-i+1} \to t_{i,3n+(l+1)n+k-i+2}\ ]_2 \\ [\ f_{i,3n+(l+1)n+k-i+1} \to f_{i,3n+(l+1)n+k-i+2}\ ]_2\end{array}\right\} \text{ for } 1 \leq i \leq k$$
$$[\ \alpha_{3n+(l+1)n+k} \to \alpha_{3n+(l+1)n+k+1}\ ]_0$$
$$[\ \beta_{3n+(l+1)n+k} \to \beta_{3n+(l+1)n+k+1}\ ]_0$$
$$[\ \gamma_{3n+(l+1)n+k} \to \gamma_{3n+(l+1)n+k+1}\ ]_2$$
$$\left.\begin{array}{l}[\ x_{i,j,3n+(l+1)n+k} \to x_{i,j,3n+(l+1)n+k+1}\ ]_2 \\ [\ \overline{x}_{i,j,3n+(l+1)n+k} \to \overline{x}_{i,j,3n+(l+1)n+k+1}\ ]_2 \\ [\ x^*_{i,j,3n+(l+1)n+k} \to x^*_{i,j,3n+(l+1)n+k+1}\ ]_2\end{array}\right\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p$$

  Therefore, the following holds

  - $\mathcal{C}_{3n+(l+1)n+k+1}(0) = \{\alpha_{3n+(l+1)n+k+1}, \beta_{3n+(l+1)n+k+1}\}$
  - In $\mathcal{C}_{3n+(l+1)n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains objects $r_{i,3n+(l+1)n+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{3n+(l+1)n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{3n+(l+1)n+k+1}(\varphi)$;
    - ⋆ an object $\gamma_{3n+(l+1)n+k+1}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and $p - l$ copies of F_i (resp. T_i)
    - ⋆ a different subset of objects $r_{i,3n+(l+1)n+k-i+2}$, $1 \leq i \leq k+1$, being $r \in \{t, f\}$.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{2np-1}$[2] holds:
  - $\mathcal{C}_{2np-1}(0) = \{\alpha_{2np-1}, \beta_{2np-1}\}$
  - In $\mathcal{C}_{2np-1}$ there are 2^n membranes labelled by 1 such that each of them contains an object $r_{n,2np}$, being $r \in \{t, f\}$.
  - In $\mathcal{C}_{n+2np-1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2np-1}(\varphi)$;
    - ⋆ an object γ_{2np-1};
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy otherwise; and
    - ⋆ a different subset of objects $r_{i,2np-i}$, $1 \leq i \leq n-1$.

[2] Note that $2np - 1 = n + 2n(p-1) + (n-1)$

Then, configuration $\mathcal{C}_{n+2np-1}$ yields $\mathcal{C}_{n+2np}$ by applying the rules:

$t_{n,2np}[\ \ ]_2 \to [\ t_{n,2np+1}\ ]_2$
$f_{n,2np}[\ \ ]_2 \to [\ f_{n,2np+1}\ ]_2$
$\left.\begin{array}{l}[\ t_{i,n+2np-i} \to t_{i,n+2np-i+1}\ ]_2 \\ [\ f_{i,n+2np-i} \to f_{i,n+2np-i}\ ]_2\end{array}\right\}$ for $1 \leq i \leq n-1$
$[\ \alpha_{n+2np-1} \to \alpha_{n+2np}\ ]_0$
$[\ \beta_{n+2np-1} \to \beta_{n+2np}\ ]_0$
$[\ \gamma_{n+2np-1} \to \gamma_{n+2np}\ ]_2$
$\left.\begin{array}{l}[\ x_{i,j,n+2np-1} \to x_{i,j,n+2np}\ ]_2 \\ [\ \overline{x}_{i,j,n+2np-1} \to \overline{x}_{i,j,n+2np}\ ]_2 \\ [\ x^*_{i,j,n+2np-1} \to x^*_{i,j,n+2np}\ ]_2\end{array}\right\}$ for $1 \leq i \leq n, 1 \leq j \leq p$

Then, we have $\mathcal{C}_{2np}(0) = \{\alpha_{2np}, \beta_{2np}\}$, and there exist 2^n empty membranes labelled by 1; and 2^n membranes labelled by 2 containing containing the input multiset $cod_{2np}(\varphi)$, an object γ_{2np}, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy otherwise and a different multiset of objects $r_{i,2np-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the truth assignment associated with the branch.

□

Proposition 3. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a) For each k $(1 \leq k \leq n-1)$ at configuration $\mathcal{C}_{2np+k}$ we have the following:
- *$\mathcal{C}_{2np+k}(0) = \{\alpha_{2np+k}, \beta_{2np+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains k objects #.*
- *there are 2^n membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset $cod_{2np+k}(\varphi)$;*
  - ⋆ *an object γ_{2np+k};*
  - ⋆ *p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+1 \leq i \leq n$; and*
  - ⋆ *objects $r_{i,2np+k-i+1}$, $k+1 \leq i \leq n$.*

(b) $\mathcal{C}_{n+2np}(0) = \{\alpha_{n+2np}, \beta_{n+2np}\}$, and in $\mathcal{C}_{n+2np}$ there are 2^n membranes labelled by 1, such that each of them contains n objects #; and 2^n membranes labelled by 2, such that each of them contains the input multiset $cod_{n+2np}(\varphi)$, an object γ_{n+2np}, p copies of every T_i and F_i, $1 \leq i \leq n$ if the truth assignment associated to the branch contains its corresponding t_i or f_i object.

Proof. (a) is going to be demonstrated by induction on k

- the base case $k = 1$ is trivial because:
  - at $\mathcal{C}_{2np}$ we have $\mathcal{C}_{2np}(0) = \{\alpha_{2np}, \beta_{2np}\}$ and there exist 2^n empty membranes labelled by 1; and 2^n membranes labelled by 2 containing the input multiset $cod_{2np}(\varphi)$, an object γ_{2np} p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if

the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy otherwise and a different multiset of objects $r_{i,2np-i+1}$, $1 \leq i \leq n$, being $r \in \{t, f\}$, that is, the truth assignment associated with the branch. Then, configuration $\mathcal{C}_{2np}$ yields $\mathcal{C}_{2np+1}$ by applying the rules.

$$\left.\begin{array}{l} [\ t_{1,2np}\ F_1\ ]_2 \to \#[\ \ ]_2 \\ [\ f_{1,2np}\ T_1\ ]_2 \to \#[\ \ ]_2 \\ [\ t_{i,2np-i+1} \to t_{i,2np-i+2}\ ]_2 \\ [\ f_{i,2np-i+1} \to f_{i,2np-i+2}\ ]_2 \end{array}\right\} \text{ for } 2 \leq i \leq n$$

$$[\ \alpha_{2np} \to \alpha_{2np+1}\ ]_0$$
$$[\ \beta_{2np} \to \beta_{2np+1}\ ]_0$$
$$[\ \gamma_{2np} \to \gamma_{2np+1}\ ]_2$$

$$\left.\begin{array}{l} [\ x_{i,j,2np} \to x_{i,j,2np+1}\ ]_2 \\ [\ \overline{x}_{i,j,2np} \to \overline{x}_{i,j,2np+1}\ ]_2 \\ [\ x^*_{i,j,2np} \to x^*_{i,j,2np+1}\ ]_2 \end{array}\right\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p$$

Thus, $\mathcal{C}_{2np+1}(0) = \{\alpha_{2np+1}, \beta_{2np+1}\}$, and there exist 2^n membranes labelled by 1 containing an object #; and 2^n membranes labelled by 2 containing the input multiset $cod_{2np+1}(\varphi)$, an object γ_{2np+1}, p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+2 \leq i \leq n$ and objects $r_{i,2np-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.

- Supposing, by induction, result is true for k $(1 \leq k \leq n-1)$
  - $\mathcal{C}_{2np+k}(0) = \{\alpha_{2np+k}, \beta_{2np+k}\}$
  - In $\mathcal{C}_{2np+k}$ there are 2^n membranes labelled by 1 such that each of them contains k objects #.
  - In $\mathcal{C}_{2np+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{2np+k}(\varphi)$;
    - ⋆ an object γ_{2np+k};
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+1 \leq i \leq n$; and
    - ⋆ objects $r_{i,2np+k-i+1}$, $k+1 \leq i \leq n$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{2np+k}$ yields configuration $\mathcal{C}_{2np+k+1}$ by applying the rules:

$$\left.\begin{array}{l} [\ t_{k+1,2np}\ F_1\ ]_2 \to \#[\ \ ]_2 \\ [\ f_{k+1,2np}\ T_1\ ]_2 \to \#[\ \ ]_2 \\ [\ t_{i,2np+k-i+1} \to t_{i,2np+k-i+2}\ ]_2 \\ [\ f_{i,2np+k-i+1} \to f_{i,2np+k-i+2}\ ]_2 \end{array}\right\} \text{ for } 2 \leq i \leq n$$

$$[\ \alpha_{2np+k} \to \alpha_{2np+k+1}\ ]_0$$
$$[\ \beta_{2np+k} \to \beta_{2np+k+1}\ ]_0$$
$$[\ \gamma_{2np+k} \to \gamma_{2np+k+1}\ ]_2$$

$$\left.\begin{array}{l} [\ x_{i,j,2np+k} \to x_{i,j,2np+k+1}\ ]_2 \\ [\ \overline{x}_{i,j,2np+k} \to \overline{x}_{i,j,2np+k+1}\ ]_2 \\ [\ x^*_{i,j,2np+k} \to x^*_{i,j,2np+k+1}\ ]_2 \end{array}\right\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p$$

Therefore, the following holds

- $\mathcal{C}_{2np+k+1}(0) = \{\alpha_{2np+k+1}, \beta_{2np+k+1}\}$
- In $\mathcal{C}_{2np+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains $k+1$ objects $\#$.
- In $\mathcal{C}_{2np+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod_{2np+k+1}(\varphi)$;
  - ⋆ an object $\gamma_{2np+k+1}$;
  - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_i (resp. T_i) if $k+2 \leq i \leq n$; and
  - ⋆ objects $r_{i,2np+k-i+2}$, $k+2 \leq i \leq n$, being $r \in \{t, f\}$.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{n+2np-1}$[3] holds:
  - $\mathcal{C}_{n+2np-1}(0) = \{\alpha_{n+2np-1}, \beta_{n+2np-1}\}$
  - In $\mathcal{C}_{n+2np-1}$ there are 2^n membranes labelled by 1 such that each of them contains $n-1$ objects $\#$.
  - In $\mathcal{C}_{n+2np-1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{n+2np-1}(\varphi)$;
    - ⋆ an object $\gamma_{n+2np-1}$;
    - ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if the corresponding t_i (resp. f_i) object exists in that branch, and 1 copy of F_n (resp. T_n); and
    - ⋆ an object $r_{n,2np}$, being $r \in \{t, f\}$.

    Then, configuration $\mathcal{C}_{n+2np-1}$ yields configuration $\mathcal{C}_{n+2np}$ by applying the rules:

$$\left.\begin{array}{l} [\ t_{n,2np}\ F_1\ ]_2 \to \#[\ \ ]_2 \\ [\ f_{n,2np}\ T_1\ ]_2 \to \#[\ \ ]_2 \\ [\ \alpha_{n+2np-1} \to \alpha_{n+2np}\ ]_0 \\ [\ \beta_{n+2np-1} \to \beta_{n+2np}\ ]_0 \\ [\ \gamma_{n+2np-1} \to \gamma_{n+2np}\ ]_2 \\ [\ x_{i,j,n+2np-1} \to x_{i,j,n+2np}\ ]_2 \\ [\ \overline{x}_{i,j,n+2np-1} \to \overline{x}_{i,j,n+2np}\ ]_2 \\ [\ x^*_{i,j,n+2np-1} \to x^*_{i,j,n+2np}\ ]_2 \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p$$

    Therefore, the following holds
  - $\mathcal{C}_{n+2np}(0) = \{\alpha_{n+2np}, \beta_{n+2np}\}$
  - In $\mathcal{C}_{n+2np}$ there are 2^n membranes labelled by 1 such that each of them contains n objects $\#$.
  - In $\mathcal{C}_{n+2np}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod_{n+2np}(\varphi)$;
    - ⋆ an object γ_{n+2np}; and

[3] Note that $n + 2np - 1 = 2np + (n-1)$

- ⋆ p copies of T_i (resp. F_i) being $1 \leq i \leq n$ if their corresponding t_i (resp. f_i) object exists in that branch.

□

5.2 First checking stage

At this stage, we try to determine the clauses satisfied for the truth assignment encoded by each branch. For that, rules from **5.5** will be applied in such manner that in the m-th step, being $m = ln+k$ ($1 \leq k \leq n$, $0 \leq l \leq p-1$), clause C_{l+1} will be evaluated with the k-th variable of the formula. This stage will take exactly np steps.

Proposition 4. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a) For each k ($1 \leq k \leq n$) and l ($0 \leq l \leq p-1$) at configuration $\mathcal{C}_{n+2np+ln+k}$ we have the following:
- *$\mathcal{C}_{n+2np+ln+k}(0) = \{\alpha_{n+2np+ln+k}, \beta_{n+2np+ln+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains*
  - ⋆ *m objects $c_{j,t}$ ($1 \leq j \leq l+1$, $0 \leq t \leq ln+k-1$), that is, clauses that have been satisfied by any variable; and*
  - ⋆ *$n+ln+k-m$ objects #.*
- *There are 2^n membranes labelled by 2 such that each of them contains*
  - ⋆ *the $(n-k)$-th last elements of $cod_{n+2np+ln+k}(\varphi)_{l+1}^{l+1}$;*
  - ⋆ *the input multiset $cod_{n+2np+ln+k}(\varphi)_{l+2}^{p}$;*
  - ⋆ *an object $\gamma_{n+2np+ln+k}$; and*
  - ⋆ *$p-l$ copies of objects T_i or F_i, $k+1 \leq i \leq n$, $p-l-1$ copies otherwise, corresponding to the truth assignment assigned to the branch.*

(b) $\mathcal{C}_{n+3np}(0) = \{\alpha_{n+3np}, \beta_{n+3np}\}$, and in $\mathcal{C}_{n+3np}$ there are 2^n membranes labelled by 1, such that each of them contains m objects $c_{j,t}$ ($1 \leq j \leq p$, $0 \leq t \leq np-1$), that is, the clauses satisfied by any variable and $n+np-m$ objects #; and 2^n membranes labelled by 2 such that each of them contains an object γ_{n+3np}.

Proof. (a) is going to be demonstrated by induction on l

- The base case $l = 0$ is goig to be demonstrated by induction on k
  - The base case $k = 1$ is trivial because:
    - at configuration $\mathcal{C}_{n+2np}$ we have: $\mathcal{C}_{n+2np}(0) = \{\alpha_{n+2np}, \beta_{n+2np}\}$ and there exist 2^n membranes labelled by 1, such that each of them contains; and 2^n membranes labelled by 2 such that each of them contains n objects # the input multiset $cod_{n+2np}(\varphi)$, an object γ_{n+2np} and p copies of objects T_i and F_i, $1 \leq i \leq n$, representing the correspondent truth assignment to the branch. Then, configuration $\mathcal{C}_{n+2np}$ yields configuration $\mathcal{C}_{n+2np+1}$ by applying the rules:

$$
\begin{array}{l}
[\ T_1\ x_{1,1,n+2np}\ ]_2 \longrightarrow c_{1,0}[\ \ ]_2 \\
[\ T_1\ \overline{x}_{1,1,n+2np}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ T_1\ x^*_{1,1,n+2np}\ ]_2 \longrightarrow \#[\ \ ]_2 \quad {}^{4} \\
[\ F_1\ x_{1,1,n+2np}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ F_1\ \overline{x}_{1,1,n+2np}\ ]_2 \longrightarrow c_{1,0}[\ \ ]_2 \\
[\ F_1\ x^*_{1,1,n+2np}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ \alpha_{n+2np} \rightarrow \alpha_{n+2np+1}\ ]_0 \\
[\ \beta_{n+2np} \rightarrow \beta_{n+2np+1}\ ]_0 \\
[\ \gamma_{n+2np} \rightarrow \gamma_{n+2np+1}\ ]_2 \\
\left.\begin{array}{l}
[\ x_{i,j,n+2np} \rightarrow x_{i,j,n+2np+1}\ ]_2 \\
[\ \overline{x}_{i,j,n+2np} \rightarrow \overline{x}_{i,j,n+2np+1}\ ]_2 \\
[\ x^*_{i,j,n+2np} \rightarrow x^*_{i,j,n+2np+1}\ ]_2
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p
\end{array}
$$

Thus, $\mathcal{C}_{n+2np+1}(0) = \{\alpha_{n+2np+1}, \beta_{n+2np+1}\}$, and there exist 2^n membranes labelled by 1 containing n objects # and an object $c_{1,0}$ if the corresponding truth assignment makes true clause 1 with variable 1, another object # otherwise; and 2^n membranes labelled by 2 containing the last $n-1$ elements of $cod_{n+2np+1}(\varphi)^1_1$, the input multiset $cod_{n+2np+1}(\varphi)^p_2$, p copies of T_i or F_i, being $2 \leq i \leq n$, and $p-1$ copies of T_1 or F_1.

- Supposing, by induction, result is true for k $(1 \leq k \leq n)$
  - $\mathcal{C}_{n+2np+k}(0) = \{\alpha_{n+2np+k}, \beta_{n+2np+k}\}$
  - In $\mathcal{C}_{n+2np+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - m objects $c_{1,t}$ $(0 \leq t \leq k-1)$, that is, the number of variables with the corresponding truth assignment that makes true the input formula φ; and
    - $n+k-m$ objects #.
  - In $\mathcal{C}_{n+2np+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - the $(n-k)$-th last elements of $cod_{n+2np+k}(\varphi)^1_1$;
    - the input multiset $cod_{n+2np+k}(\varphi)^p_2$;
    - an object $\gamma_{n+2np+k}$; and
    - p copies of objects T_i or F_i, $k+1 \leq i \leq n$, $p-1$ copies if $1 \leq i \leq k$, corresponding to the truth assignment assigned to the branch.

    Then, configuration $\mathcal{C}_{n+2np+k}$ yields configuration $\mathcal{C}_{n+2np+k+1}$ by applying the rules:

$$
\begin{array}{l}
[\ T_k\ x_{1,1,n+2np+k}\ ]_2 \longrightarrow c_{1,0}[\ \ ]_2 \\
[\ T_k\ \overline{x}_{1,1,n+2np+k}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ T_k\ x^*_{1,1,n+2np+k}\ ]_2 \longrightarrow \#[\ \ ]_2 \quad {}^{5} \\
[\ F_k\ x_{1,1,n+2np+k}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ F_k\ \overline{x}_{1,1,n+2np+k}\ ]_2 \longrightarrow c_{1,0}[\ \ ]_2 \\
[\ F_k\ x^*_{1,1,n+2np+k}\ ]_2 \longrightarrow \#[\ \ ]_2
\end{array}
$$

[4] If $k = 1, l = 0$, then $i = 1, j = 1$, so $2np + n + n(j-1) + (i-1) = n + 2np$

[5] If $l = 0$, then $i = k+1, j = 1$, so $2np + 2n + n(j-1) + (i-1) = 2n + 2np + k$

$$
\begin{array}{l}
[\ \alpha_{n+2np+k} \rightarrow \alpha_{n+2np+k+1}\ ]_0 \\
[\ \beta_{n+2np+k} \rightarrow \beta_{n+2np+k+1}\ ]_0 \\
[\ \gamma_{n+2np+k} \rightarrow \gamma_{n+2np+k+1}\ ]_2 \\
\left.\begin{array}{l}
[\ x_{i,j,n+2np+k} \rightarrow x_{i,j,n+2np+k+1}\ ]_2 \\
[\ \overline{x}_{i,j,n+2np+k} \rightarrow \overline{x}_{i,j,n+2np+k+1}\ ]_2 \\
[\ x^*_{i,j,n+2np+k} \rightarrow x^*_{i,j,n+2np+k+1}\ ]_2
\end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j \leq p \\
[\ c_{1,t} \rightarrow c_{1,t+1}\ ]_1 \text{ for } 0 \leq t \leq k-1
\end{array}
$$

Therefore, the following holds

- $\mathcal{C}_{n+2np+k+1} = \{\alpha_{n+2np+k+1}, \beta_{n+2np+k+1}\}$
- In $\mathcal{C}_{n+2np+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - $\star$ m objects $c_{1,t}$ $(0 \leq t \leq k)$, that is, the number of variables with the corresponding truth assignment that makes true the clause $\mathcal{C}_1$; and
  - $\star$ $n + k + 1 - m$ objects $\#$.
- In $\mathcal{C}_{n+2np+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - $\star$ the $(n-k+1)$-th last elements of $cod_{n+2np+k+1}(\varphi)^1_1$;
  - $\star$ the input multiset $cod_{n+2np+k+1}(\varphi)^p_2$,
  - $\star$ an object $\gamma_{n+2np+k+1}$; and
  - $\star$ p copies of objects T_i or F_i, $k+2 \leq i \leq n$, $p-1$ copies if $1 \leq i \leq k+1$, corresponding to the truth assignment assigned to the branch.

- Supposing, by induction, result is true for l $(0 \leq l \leq p-1)$
  - The base case $k = 1$ is trivial because:
    - at configuration $\mathcal{C}_{n+2np+(l+1)n}$ we have: $\mathcal{C}_{n+2np+(l+1)n}(0) = \{\alpha_{n+2np+(l+1)n}, \beta_{n+2np+(l+1)n}\}$ and there exist 2^n membranes labelled by 1 containing m objects $c_{j,t}$ $(1 \leq j \leq l,\ 0 \leq t \leq ln-1)$, that is, the number of variables with the corresponding truth assignment that makes true the clauses from $\mathcal{C}_1$ to $\mathcal{C}_l$ and $n + (l+1)n - m$ objects $\#$; and 2^n membranes labelled by 2 containing the input multiset $cod_{n+2np+(l+1)n}(\varphi)^p_{l+1}$, an object $\gamma_{n+2np+(l+1)n}$ and $p - l$ copies of objects T_i or F_i, $1 \leq i \leq n$. Then, configuration $\mathcal{C}_{n+2np+(l+1)n}$ yields configuration $\mathcal{C}_{n+2np+(l+1)n+1}$ by applying the rules:

$$
\begin{array}{l}
[\ T_1\ x_{1,1,n+2np+(l+1)n}\ ]_2 \longrightarrow c_{l+1,0}[\ \ ]_2 \\
[\ T_1\ \overline{x}_{1,1,n+2np+(l+1)n}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ T_1\ x^*_{1,1,n+2np+(l+1)n}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ F_1\ x_{1,1,n+2np+(l+1)n}\ ]_2 \longrightarrow c_{l+1,0}[\ \ ]_2 \\
[\ F_1\ \overline{x}_{1,1,n+2np+(l+1)n}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ F_1\ x^*_{1,1,n+2np+(l+1)n}\ ]_2 \longrightarrow \#[\ \ ]_2 \\
[\ \alpha_{n+2np+(l+1)n} \rightarrow \alpha_{n+2np+(l+1)n+1}\ ]_0 \\
[\ \beta_{n+2np+(l+1)n} \rightarrow \beta_{n+2np+(l+1)n+1}\ ]_0 \\
[\ \gamma_{n+2np+(l+1)n} \rightarrow \gamma_{n+2np+(l+1)n+1}\ ]_2
\end{array}
$$

$$\left.\begin{array}{l}[\ x_{i,j,n+2np+(l+1)n} \to x_{i,j,n+2np+(l+1)n+1}\ ]_2\\ [\ \overline{x}_{i,j,n+2np+(l+1)n} \to \overline{x}_{i,j,n+2np+(l+1)n+1}\ ]_2\\ [\ x^*_{i,j,n+2np+(l+1)n} \to x^*_{i,j,n+2np+(l+1)n+1}\ ]_2\end{array}\right\} \text{for } \begin{array}{l}1 \le i \le n\\ 1 \le j \le p\end{array}$$

$[\ c_{j,t} \to c_{1,t+1}\ ]_1$ for $1 \le j \le l+1, 0 \le t \le ln-1$

Thus, $\mathcal{C}_{n+2np+(l+1)n+1}(0) = \{\alpha_{n+2np+(l+1)n+1}, \beta_{n+2np+(l+1)n+1}\}$, and there exist 2^n membranes labelled by 1 containing m objects $c_{j,t}$ ($1 \le j \le l+1$, $0 \le t \le ln$), that is, the number of variables with the corresponding truth assignment that makes true the clauses from $\mathcal{C}_1$ to $\mathcal{C}_{l+1}$ and $n+(l+1)n+1-m$ objects #; and 2^n membranes labelled by 2 containing the last $n-1$ elements of $cod_{n+2np+(l+1)n+1}(\varphi)^{l+1}_{l+1}$, the input multiset $cod_{n+2np+(l+1)n+1}(\varphi)^{p}_{l+2}$, $p-l$ copies of T_i or F_i, being $2 \le i \le n$, and $p-l-1$ copies of T_1 or F_1.

- Supposing, by induction, result is true for k ($1 \le k \le n$)
  - $\mathcal{C}_{n+2np+(l+1)n+k}(0) = \{\alpha_{n+2np+(l+1)n+k}, \beta_{n+2np+(l+1)n+k}\}$
  - In $\mathcal{C}_{n+2np+(l+1)n+k}$ there are 2^n membranes labelled by 1 such that each of them contains
    - $\star$ m objects $c_{j,t}$ ($1 \le j \le l+1$, $0 \le t \le ln+k-1$), that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_{l+1}$; and
    - $\star$ $n+(l+1)n+k+1-m$ objects #.
  - In $\mathcal{C}_{n+2np+(l+1)n+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - $\star$ the $(n-k)$-th last elements of $cod_{n+2np+(l+1)n+k}(\varphi)^{l+1}_{l+1}$;
    - $\star$ the input multiset $cod_{n+2np+(l+1)n+k}(\varphi)^{p}_{l+2}$,
    - $\star$ an object $\gamma_{n+2np+(l+1)n+k}$; and
    - $\star$ $p-l$ copies of objects T_i or F_i, $k+1 \le i \le n$, $p-l-1$ copies if $1 \le i \le k$, corresponding to the truth assignment assigned to the branch.

    Then, configuration $\mathcal{C}_{n+2np+(l+1)n+k}$ yields configuration $\mathcal{C}_{n+2np+(l+1)n+k+1}$ by applying the rules:

$$[\ T_k\ x_{1,1,n+2np+(l+1)n+k}\ ]_2 \longrightarrow c_{l+1}[\ \ ]_2$$
$$[\ T_k\ \overline{x}_{1,1,n+2np+(l+1)n+k}\ ]_2 \longrightarrow \#[\ \ ]_2$$
$$[\ T_k\ x^*_{1,1,n+2np+(l+1)n+k}\ ]_2 \longrightarrow \#[\ \ ]_2$$
$$[\ F_k\ x_{1,1,n+2np+(l+1)n+k}\ ]_2 \longrightarrow \#[\ \ ]_2$$
$$[\ F_k\ \overline{x}_{1,1,n+2np+(l+1)n+k}\ ]_2 \longrightarrow c_{l+1}[\ \ ]_2$$
$$[\ F_k\ x^*_{1,1,n+2np+(l+1)n+k}\ ]_2 \longrightarrow \#[\ \ ]_2$$
$$[\ \alpha_{n+2np+(l+1)n+k} \to \alpha_{n+2np+(l+1)n+k+1}\ ]_0$$
$$[\ \beta_{n+2np+(l+1)n+k} \to \beta_{n+2np+(l+1)n+k+1}\ ]_0$$
$$[\ \gamma_{n+2np+(l+1)n+k} \to \gamma_{n+2np+(l+1)n+k+1}\ ]_2$$
$$\left.\begin{array}{l}[\ x_{i,j,n+2np+(l+1)n+k} \to x_{i,j,n+2np+(l+1)n+k+1}\ ]_2\\ [\ \overline{x}_{i,j,n+2np+(l+1)n+k} \to \overline{x}_{i,j,n+2np+(l+1)n+k+1}\ ]_2\\ [\ x^*_{i,j,n+2np+(l+1)n+k} \to x^*_{i,j,n+2np+(l+1)n+k+1}\ ]_2\end{array}\right\} \text{for } \begin{array}{l}1 \le i \le n\\ 1 \le j \le p\end{array}$$

$[\ c_{j,t} \to c_{j,t+1}\ ]_1$ for $1 \le j \le l+1, 0 \le t \le ln+k-1$

Therefore, the following holds

- $\mathcal{C}_{n+2np+(l+1)n+k+1}(0) = \{\alpha_{n+2np+(l+1)n+k+1}, \beta_{n+2np+(l+1)n+k+1}\}$
- In $\mathcal{C}_{n+2np+(l+1)n+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - $\star$ m objects $c_{j,t}$ $(1 \leq j \leq l+1, 0 \leq t \leq ln+k)$, that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_{l+1}$; and
  - $\star$ $n + (l+1)n + k + 1 - m$ objects $\#$.
- In $\mathcal{C}_{n+2np+(l+1)n+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - $\star$ the $(n-(k+1))$-th last elements of $cod_{n+2np+(l+1)n+k+1}(\varphi)^{l+1}_{l+1}$,
  - $\star$ the input multiset $cod_{n+2np+(l+1)n+k+1}(\varphi)^{p}_{l+1}$,
  - $\star$ an object $\gamma_{n+2np+(l+1)n+k+1}$;
  - $\star$ $p-l$ copies of objects T_i or F_i, $k+2 \leq i \leq n$, $p-l-1$ copies if $1 \leq i \leq k+1$, corresponding to the truth assignment assigned to the branch.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{n+3np-1}$ [6] holds:
  - $\mathcal{C}_{n+3np-1}(0) = \{\alpha_{n+3np-1}, \beta_{n+3np-1}\}$
  - In $\mathcal{C}_{n+3np-1}$ there are 2^n membranes labelled by 1 such that each of them contains
    - $\star$ m objects $c_{j,t}$ $(1 \leq j \leq p, 0 \leq t \leq np-2)$, that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_p$; and
    - $\star$ $n + np - 1 - m$ objects $\#$.
  - In $\mathcal{C}_{n+3np-1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - $\star$ the last element of $cod_{n+3np-1}(\varphi)^{p}_{p}$;
    - $\star$ an object $\gamma_{n+3np-1}$; and
    - $\star$ an object T_n or F_n corresponding to the truth assignment assigned to the branch.

  Then, configuration $\mathcal{C}_{n+3np-1}$ yields $\mathcal{C}_{n+3np}$ by applying the rules:

  $[\ T_n\ x_{n,p,n+3np-1}\ ]_2 \longrightarrow c_{p,0}[\ \ ]_2$
  $[\ T_n\ \overline{x}_{n,p,n+3np-1}\ ]_2 \longrightarrow \#[\ \ ]_2$
  $[\ T_n\ x^*_{n,p,n+3np-1}\ ]_2 \longrightarrow \#[\ \ ]_2$
  $[\ F_n\ x_{n,p,n+3np-1}\ ]_2 \longrightarrow \#[\ \ ]_2$
  $[\ F_n\ \overline{x}_{n,p,n+3np-1}\ ]_2 \longrightarrow c_{p,0}[\ \ ]_2$
  $[\ F_n\ x^*_{n,p,n+3np-1}\ ]_2 \longrightarrow \#[\ \ ]_2$
  $[\ \alpha_{n+2np+(l+1)n+k} \rightarrow \alpha_{n+2np+(l+1)n+k+1}\ ]_0$
  $[\ \beta_{n+2np+(l+1)n+k} \rightarrow \beta_{n+2np+(l+1)n+k+1}\ ]_0$
  $[\ \gamma_{n+2np+(l+1)n+k} \rightarrow \gamma_{n+2np+(l+1)n+k+1}\ ]_2$

[6] Note that $n + 3np - 1 = n + 3n(p-1) + (n-1)$

$$\left.\begin{array}{l}[\ x_{i,j,n+2np+(l+1)n+k} \to x_{i,j,n+2np+(l+1)n+k+1}\ ]_2 \\ [\ \overline{x}_{i,j,n+2np+(l+1)n+k} \to \overline{x}_{i,j,n+2np+(l+1)n+k+1}\ ]_2 \\ [\ x^*_{i,j,n+2np+(l+1)n+k} \to x^*_{i,j,n+2np+(l+1)n+k+1}\ ]_2\end{array}\right\} \text{for } \begin{array}{l}1 \le i \le n \\ 1 \le j \le p\end{array}$$

$[\ c_{j,t} \to c_{j,t+1}\ ]_2$ for $1 \le j \le l+1, 0 \le t \le ln+k-1$

Therefore, the following holds

- $\mathcal{C}_{n+3np}(0) = \{\alpha_{n+3np}, \beta_{n+3np}\}$
- In $\mathcal{C}_{n+3np}$ there are 2^n membranes labelled by 1 such that each of them contains
  - ⋆ m objects $c_{j,t}$ $(1 \le j \le p,\ 0 \le t \le np-1)$, that is, the number of variables with the corresponding truth assignment that makes true clauses from $\mathcal{C}_1$ to $\mathcal{C}_p$; and
  - ⋆ $n + np - m$ objects #.
- In $\mathcal{C}_{n+3np}$ there are 2^n membranes labelled by 2 such that each of them contains an object γ_{n+3np}.

□

5.3 Second checking stage

At this stage, started at configuration $\mathcal{C}_{n+3np}$, we try to determine the truth assignments that make true the input formula φ, using rules from **5.6**. We are going to divide this stage in two phases. The first one will be devoted to send in all the objects c_j, for $1 \le j \le p$ in order to get them ready for the next phase.

Let $k = ln + i$ $(0 \le l \le p-1, 1 \le i \le n)$, so we can refer to each clause $(l-1)$ when we are doing the verification. Let $m = \sum_{j=1}^{p} m_j$, being m_j the number of objects $c_{j,k}$ in each membrane 1 at step $\mathcal{C}_{n+3np}$. In this stage, we cannot be sure of how many objects $c_{l+1,k}$ are present at each membrane when $i \neq 0$ [7], so if we cannot be sure of that, we are going to say that there are $\widetilde{m}_j$ (remember that $\widetilde{m}_j$ is always less than or equal to m_j) objects within membrane 1. We will ignore objects # since they have no effect from here.

Proposition 5. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a) For each k $(1 \le k \le np-1)$ at configuration $\mathcal{C}_{n+3np+k}$ we have the following:

- *$\mathcal{C}_{n+3np+k}(0) = \{\alpha_{n+3np+k}, \beta_{n+3np+k}\}$*
- *There are 2^n membranes labelled by 1 such that each of them contains $\widetilde{m}_{l+1}$ objects $c_{l+1,t}$ $((p-1)n+1 \le t \le np-1)$ and m_j objects $c_{j,t}$ $(l+2 \le j \le p, ln+i \le t \le np-1)$*
- *There are 2^n membranes labelled by 2 such that each of them contains*
  - ⋆ *an object $\gamma_{n+3np+k}$; and*
  - ⋆ *m_j objects c_j for $1 \le j \le l$ and $m_{l+1} - \widetilde{m}_{l+1}$ objects c_{l+1}*

[7] That is because objects $c_{j,k}$ do not have to be created consecutively.

(*b*) $\mathcal{C}_{n+4np}(0) = \{\alpha_{n+4np}, \beta_{n+4np}\}$, *there are* 2^n *empty membranes labelled by 1; and* 2^n *membranes labelled by 2, such that each of them contains* m *objects* c_j $(1 \leq j \leq p)$ *and an object* γ_{n+4np}.

Proof. (*a*) is going to be demonstrated by induction on k

- The base case $k = 1$ is trivial because: At configuration $\mathcal{C}_{n+3np}$ we have: $\mathcal{C}_{n+3np}(0) = \{\alpha_{n+3np}, \beta_{n+3np}\}$ and there exist 2^n membranes labelled by 1 containing m objects $c_{j,t}$ $(1 \leq j \leq k, 0 \leq t \leq np-1)$; and 2^n membranes labelled by 2 containing an object γ_{n+3np}. Then, configuration $\mathcal{C}_{n+3np}$ yields configuration $\mathcal{C}_{n+3np+1}$ by applying the rules:
  $[\ \alpha_{n+3np} \to \alpha_{n+3np+1}\ ]_0$
  $[\ \beta_{n+3np} \to \beta_{n+3np+1}\ ]_0$
  $[\ \gamma_{n+3np} \to \gamma_{n+3np+1}\ ]_2$
  $[c_{j,t} \longrightarrow c_{j,t+1}]_1$, for $1 \leq j \leq p$, $0 \leq k \leq np-2$
  $c_{1,np-1}[\ \ ]_2 \longrightarrow [c_1]_2$

  Thus, $\mathcal{C}_{n+3np+1}(0) = \{\alpha_{n+3np+1}, \beta_{n+3np+1}\}$, and there exist 2^n membranes labelled by 2 containing $\widetilde{m}_1$ objects c_1 and m_j objects c_j $(2 \leq j \leq p)$; and 2^n membranes labelled by 1 containing an object $\gamma_{n+3np+1}$ and $m_1 - \widetilde{m}_1$ objects c_1 [8]. Hence, the result holds for $k = 1$.
- Supposing, by induction, result is true for k $(1 \leq k \leq np-1)$
  - $\mathcal{C}_{n+3np+k}(0) = \{\alpha_{n+3np+k}, \beta_{n+3np+k}\}$
  - In $\mathcal{C}_{n+3np+k}$ there are 2^n membranes labelled by 1 such that each of them contains $\widetilde{m}_{l+1}$ objects $c_{l+1,t}$ $((p-1)n+1 \leq t \leq np-1)$ and m_j objects $c_{j,t}$ $(l+2 \leq j \leq p, ln+i \leq t \leq np-1)$.
  - In $\mathcal{C}_{n+3np+k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ an object $\gamma_{n+3np+k}$; and
    - ⋆ m_j objects c_j for $1 \leq j \leq l$ and $m_{l+1} - \widetilde{m}_{l+1}$ objects c_{l+1}.

    Then, configuration $\mathcal{C}_{n+3np+k}$ yields configuration $\mathcal{C}_{n+3np+k}$ by applying the rules:
    $[\ \alpha_{n+3np+k} \to \alpha_{n+3np+k+1}\ ]_0$
    $[\ \beta_{n+3np+k} \to \beta_{n+3np+k+1}\ ]_0$
    $[\ \gamma_{n+3np+k} \to \gamma_{n+3np+k+1}\ ]_2$
    $[c_{j,t} \longrightarrow c_{j,t+1}]_1$, for $l+1 \leq j \leq p$, $0 \leq k \leq np-2$
    $c_{l+1,np-1}[\ \ ]_2 \longrightarrow [c_{l+1}]_2$

    Therefore, the following holds
  - $\mathcal{C}_{n+3np+k+1}(0) = \{\alpha_{n+3np+k+1}, \beta_{n+3np+k+1}\}$
  - In $\mathcal{C}_{n+3np+k+1}$ there are 2^n membranes labelled by 1 such that each of them contains $\widetilde{m}_{l+1}$ objects $c_{l+1,t+1}$ $((p-1)n+1 \leq t \leq np-1)$ and m_j objects $c_{j,t+1}$ $(l+2 \leq j \leq p, ln+i \leq t \leq np-1)$.

[8] That is, if the truth assignment of variable 1 made true clause 1, then an object $c_{1,0}$ were created at $(2n+2np+1)$-th step, and it is going to be sent to the corresponding membrane 2. So, $m_1 - \widetilde{m}_1$ can be 0 or 1 in this step.

- In $\mathcal{C}_{n+3np+k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ an object $\gamma_{n+3np+k+1}$; and
  - ⋆ m_j objects c_j for $1 \leq j \leq l$ and $m_{l+1} - \widetilde{m}_{l+1}$ objects c_{l+1}.

  Hence, the result holds for $k+1$.

- In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{n+4np-1}$ holds:
  - $\mathcal{C}_{n+4np-1}(0) = \{\alpha_{n+4np-1}, \beta_{n+4np-1}\}$
  - In $\mathcal{C}_{n+4np-1}$ there are 2^n membranes labelled by 1 such that each of them contains
  - In $\mathcal{C}_{n+4np-1}$ there are 2^n membranes labelled by 2 such that each of them contains $\widetilde{m}_p$ objects $c_{p,np}$.
    - ⋆ an object $\gamma_{n+4np-1}$; and
    - ⋆ m_j objects c_j for $1 \leq j \leq p-1$ and $m_p - \widetilde{m}_p$ [9] objects c_p.

    Then, configuration $\mathcal{C}_{n+4np-1}$ yields configuration $\mathcal{C}_{n+4np}$ by applying the rules:

    $[\ \alpha_{n+4np-1} \to \alpha_{n+4np}\ ]_0$
    $[\ \beta_{n+4np-1} \to \beta_{n+4np}\ ]_0$
    $[\ \gamma_{n+4np-1} \to \gamma_{n+4np}\ ]_2$
    $c_{p,np}[\ \ ]_2 \longrightarrow [c_p]_2$

    Then, we have $\mathcal{C}_{n+4np}(0) = \{\alpha_{n+4np}, \beta_{n+4np}\}$, and there exist 2^n empty membranes labelled by 1; and there exist 2^n membranes labelled by 2 containing an object γ_{n+4np} and m objects c_j $(1 \leq j \leq p)$.

□

When objects c_j are within the membranes labelled by 2, we can start to check if all the clauses of the input formula φ are satisfied by any truth assignment. As we use objects c_j to denote that clause C_j has been satisfied by some variable, it can be possible that some c_j are missing, that is, that for some j, $1 \leq j \leq p$, c_j does not appear in any membrane labelled by 2 in $\mathcal{C}_{2n+4np}$. Let $\widetilde{j}$ be the index j [10] of that clause. It is going to take $2p$ steps.

Proposition 6. *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ be a computation of the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.*

(a_0) *For each $2k+1$ $(0 \leq k \leq p-1)$ at configuration $\mathcal{C}_{n+4np+2k+1}$ we have the following:*

- $\mathcal{C}_{n+4np+2k+1}(0) = \{\alpha_{n+4np+2k+1}, \beta_{n+4np+2k+1}\}$
- *There are 2^n membranes labelled by 1 such that each of them contains an object d_{k+1} if and only if the truth assignment associated to the branch makes true the first $k+1$ clauses.*
- *There are 2^n membranes labelled by 2 such that each of them contains*

[9] In this case, $\widetilde{m}_p$ can only take two values: 0 or 1.

[10] If $\widetilde{j}$ is not defined, we are going to suposse that it is equal to $p+1$.

  - ⋆ *an object* γ_{n+4np} *or* $d_{\widetilde{j}-1}$ *(respectively, an object* d_k*) if the corresponding truth assignment does not make true (resp., makes true) the clause* C_1 *or* C_j $(2 \leq j \leq p)$ *(resp., the first* k *clauses); and*
  - ⋆ $m_j - 1$ *objects* c_j *for* $1 \leq j \leq min(\widetilde{j}, k+1)$ *and* m_j *objects* c_j *for* $min(\widetilde{j}, k+2) \leq j \leq p$.

(a_1) *For each* $2k$ $(1 \leq k \leq p-2)$ *at configuration* $\mathcal{C}_{n+4np+2k}$ *we have the following:*

- $\mathcal{C}_{n+4np+2k}(0) = \{\alpha_{n+4np+2k}, \beta_{n+4np+2k}\}$
- *There are* 2^n *empty membranes labelled by 1.*
- *There are* 2^n *empty membranes labelled by 2 such that each of them contains*
  - ⋆ *an object* γ_{n+4np} *or* $d_{\widetilde{j}-1}$ *if the corresponding truth assignment does not make true the clause* C_1 *or* C_j $(2 \leq j \leq p)$*; and*
  - ⋆ $m_j - 1$ *objects* c_j *for* $1 \leq j \leq min(\widetilde{j}, k)$ *and* m_j *objects* c_j *for* $min(\widetilde{j}, k+1) \leq j \leq p$.

(b) $\mathcal{C}_{n+4np+2p-1}(0) = \{\alpha_{n+4np+2p-1}, \beta_{n+4np+2p-1}\}$*, and in* $\mathcal{C}_{n+4np+2p-1}$ *there are* 2^n *membranes labelled by 1, such that each of them contains an object* d_p *if and only if the corresponding truth assignment makes true the input formula* φ *(*$d_{\widetilde{j}-1}$ *otherwise); and* 2^n *membranes labelled by 2, such that each of them contains* $m_j - 1$ *objects* c_j *for* $1 \leq j \leq min(\widetilde{j}, p+1)$*,* m_j *objects* c_j *for* $min(\widetilde{j}, p+1) \leq j \leq p$ *and an object* γ_{n+4np} *(respectively,* $d_{\widetilde{j}}$*) if clause* C_1 *(resp.,* C_j*) is not satisfied by the corresponding truth assignment.*

Proof. (a) is going to be demonstrated by induction on k

- The base case $k = 1$ is trivial because:

(a_0) at configuration $\mathcal{C}_{n+4np}$ we have: $\mathcal{C}_{n+4np}(0) = \{\alpha_{n+4np}, \beta_{n+4np}\}$ and there exist 2^n empty membranes labelled by 1; and there exist 2^n membranes labelled by 2 containing an object γ_{n+4np} and m objects c_j $(1 \leq j \leq p)$. Then, configuration $\mathcal{C}_{n+4np}$ yields configuration $\mathcal{C}_{n+4np+1}$ by applying the rules:

$$[\ \alpha_{n+4np} \to \alpha_{n+4np+1}\ ]_0$$
$$[\ \beta_{n+4np} \to \beta_{n+4np+1}\ ]_0$$
$$[\ \gamma_{4np+2n}\ c_1\ ]_2 \longrightarrow d_1[\ \ ]_2$$

(a_1) at $\mathcal{C}_{n+4np+1}(0) = \{\alpha_{n+4np+1}, \beta_{n+4np+1}\}$ and there exist 2^n membranes labelled by 1 containing an object d_1 if and only if there was at least one object c_1 within membrane labelled by 1 at configuration $\mathcal{C}_{n+4np}$; and 2^n membranes labelled by 2 containing an object γ_{n+4np} if and only if there were no objects c_1 at configuration $\mathcal{C}_{n+4np}$, $m_1 - 1$ (respectively, m_1) objects c_1 if there was any object c_j in this membrane in the previous configuration (resp., m_1) and m_j objects c_j for $2 \leq j \leq p$. Then, the configuration $\mathcal{C}_{n+4np+1}$ yields configuration $\mathcal{C}_{n+4np+2}$ by applying the rules:

$$[\ \alpha_{n+4np+1} \to \alpha_{n+4np+2}\ ]_0$$
$$[\ \beta_{n+4np+1} \to \beta_{n+4np+2}\ ]_0$$
$$d_1\ [\ \ ]_2 \longrightarrow [\ d_1\ ]_2$$

Thus, $\mathcal{C}_{n+4np+2}(0) = \{\alpha_{n+4np+2}, \beta_{n+4np+2}\}$, and there exist 2^n empty membranes labelled by 1; and there exist 2^n membranes labelled by 2

containing an object d_1 (respectively, γ_{n+4np}) if the corresponding truth assignment makes true (resp., doesn't make true) clause C_1, $m_1 - 1$ (resp., m_1) objects c_1 and m_j objects c_j for $1 \leq j \leq p$. Hence, the result holds for $k = 1$.

- Supposing, by induction, result is true for k $(0 \leq k \leq p-1)$
  - $\mathcal{C}_{n+4np+2k}(0) = \{\alpha_{n+4np+2k}, \beta_{n+4np+2k}\}$
  - In $\mathcal{C}_{n+4np+2k}$ there are 2^n empty membranes labelled by 1.
  - In $\mathcal{C}_{n+4np+2k}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ an object γ_{n+4np} or $d_{\widetilde{j-1}}$ (respectively, an object d_k) if the corresponding truth assignment does not make true (resp., makes true) the clause C_1 or C_j $(2 \leq j \leq p)$ (resp., the first k clauses); and
    - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k+1)$ and m_j objects c_j for $min(\widetilde{j}, k+2) \leq j \leq p$.

    Then, configuration $\mathcal{C}_{n+4np+2k}$ yields configuration $\mathcal{C}_{n+4np+2k+1}$ by applying the rules:

    $[\ \alpha_{n+4np+2k} \to \alpha_{n+4np+2k+1}\ ]_0$
    $[\ \beta_{n+4np+2k} \to \beta_{n+4np+2k+1}\ ]_0$
    $[\ d_k\ c_{k+1}\ ]_2 \longrightarrow d_{k+1}[\ \ ]_2$

    Therefore, the following holds
  - $\mathcal{C}_{n+4np+2k+1}(0) = \{\alpha_{n+4np+2k+1}, \beta_{n+4np+2k+1}\}$
  - In $\mathcal{C}_{n+4np+2k+1}$ there are 2^n membranes labelled by 1 such that each of them contains an object d_{k+1} if and only if the corresponding truth assignment makes true the first $k+1$ clauses.
  - In $\mathcal{C}_{n+4np+2k+1}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ an object γ_{n+4np} or $d_{\widetilde{j-1}}$ if the corresponding truth assignment does not make true the clause C_1 or C_j $(2 \leq j \leq p)$; and
    - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k)$ and m_j objects c_j for $min(\widetilde{j}, k+1) \leq j \leq p$.

    Then, configuration $\mathcal{C}_{n+4np+2k+1}$ yields $\mathcal{C}_{n+4np+2k+2}$ by applying the rules:

    $[\ \alpha_{n+4np+2k+1} \to \alpha_{n+4np+2k+2}\ ]_0$
    $[\ \beta_{n+4np+2k+1} \to \beta_{n+4np+2k+2}\ ]_0$
    $d_{k+1}[\ \ ]_2 \longrightarrow [\ d_{k+1}\ ]_2$

    Therefore, the following holds
  - $\mathcal{C}_{n+4np+2k+2}(0) = \{\alpha_{n+4np+2k+2}, \beta_{n+4np+2k+2}\}$
  - In $\mathcal{C}_{n+4np+2k+2}$ there are 2^n empty membranes labelled by 1.
  - In $\mathcal{C}_{n+4np+2k+2}$ there are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ an object γ_{n+4np} or $d_{\widetilde{j-1}}$ (respectively, an object d_{k+1}) if the corresponding truth assignment does not make true (resp., makes true) the clause C_1 or C_j $(2 \leq j \leq p)$ (resp., the first $k+1$ clauses); and

  - ⋆ $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, k+2)$ and m_j objects c_j for $min(\widetilde{j}, k+3) \leq j \leq p$.

  Hence, the result holds for $k+1$.

- In order to prove (b) it is enough to notice that, on the one han, from (a) configuration $\mathcal{C}_{n+4np+2p-2}$ holds:
  - $\mathcal{C}_{n+4np+2p-2}(0) = \{\alpha_{n+4np+2p-2}, \beta_{n+4np+2p-2}\}$
  - In $\mathcal{C}_{n+4np+2p-2}$ there are 2^n empty membranes labelled by 1.
  - In $\mathcal{C}_{n+4np+2p-2}$ there are 2^n membranes labelled by 2 such that each of them contains
    - an object γ_{n+4np} or $d_{\widetilde{j}-1}$ (respectively, d_{p-1}) if the corresponding truth assignment does not make true the clause C_1 or C_j $(2 \leq j \leq p-1)$ (resp., makes true clauses C_j $(1 \leq j \leq p-1)$); and
    - $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, p-1)$ and m_j objects c_j for $min(\widetilde{j}, p) \leq j \leq p$

    Then, configuration $\mathcal{C}_{n+4np+2p-2}$ yields configuration $\mathcal{C}_{n+4np+2p-1}$ by applying the rules:

    $[\ \alpha_{n+4np+2p-2} \to \alpha_{n+4np+2p-1}\ ]_0$
    $[\ \beta_{n+4np+2p-2} \to \beta_{n+4np+2p-1}\ ]_0$
    $[\ d_{p-1}\ c_p\ ]_2 \longrightarrow d_p\ [\ \ ]_2$

    Then, we have $\mathcal{C}_{n+4np+2p-1}(0) = \{\alpha_{n+4np+2p-1}, \beta_{n+4np+2p-1}\}$, and in $\mathcal{C}_{n+4np+2p-1}$ there are 2^n membranes labelled by 1, such that each of them contains an object d_p if and only if the corresponding truth assignment makes true the input formula φ ($d_{\widetilde{j}-1}$ otherwise); and 2^n membranes labelled by 2, such that each of them contains $m_j - 1$ objects c_j for $1 \leq j \leq min(\widetilde{j}, p+1)$, m_j objects c_j for $min(\widetilde{j}, p+1) \leq j \leq p$ and an object γ_{n+4np} (respectively, $d_{\widetilde{j}}$) if clause C_1 (resp., C_j) is not satisfied by the corresponding truth assignment.

□

5.4 Output stage

The output phase starts at configuration $\mathcal{C}_{n+4np+2p-1}$, and takes exactly two steps when there is an affirmative answer and three steps when there is a negative one. Rules from **5.7** are devoted to compute this stage.

- *Affirmative answer*: In this case, at configuration $\mathcal{C}_{n+4np+2p-1}$, in some membrane 1 there is an object d_p. By applying the rule $[\ d_p\ ]_1 \longrightarrow d_p[\ \ ]_1$ (at the same time that $[\ \alpha_{n+4np+2p-1} \to \alpha_{n+4np+2p}\ ]_0$ and $[\ \beta_{n+4np+2p-1} \to \beta_{n+4np+2p}\ ]_0$ are executed), an object d_p is produced in membrane 0. Then by applying the rules $[\ \alpha_{4np+n+2p}\ d_p\ ]_0 \longrightarrow \texttt{yes}[\ \ ]_0$ and $[\ \beta_{n+4np+2p} \to \beta_{n+4np+2p+1}\ ]_0$, an object `yes` is released to environment and the computation halts.

- *Negative answer*: In this case, at configuration $\mathcal{C}_{n+4np+2p-1}$, there are no membranes labelled by 1 that contains an object d_p, so the only rules executed are $[\ \alpha_{n+4np+2p-1} \to \alpha_{n+4np+2p}\ ]_0$ and $[\ \beta_{n+4np+2p-1} \to \beta_{n+4np+2p}\ ]_0$. Rule $[\ \beta_{n+4np+2p} \to \beta_{n+4np+2p+1}\ ]_0$ is executed in the next step. Thus, at configuration $\mathcal{C}_{n+4np+2p+1}$ in membrane labelled by 0 we execute have a copy of object $\alpha_{n+4np+2p}$ and a copy of object $\beta_{n+4np+2p+1}$. By applying the rule $[\ \alpha_{4np+n+2p}\ \beta_{4np+n+2p+1}]_0 \longrightarrow \texttt{no}[\ \ ]_0$ an object `no` is released to the environment and then the computation halts.

5.5 Result

Theorem 1. $\texttt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0(+e_s,mcmp_{out},-d,+n)}$.

Proof. The family $\mathbf{\Pi}$ of P systems previously constructed verifies the following:

(a) The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines because for each $n, p \in \mathbb{N}$, the rules of $\Pi(\langle n, p\rangle)$ of the family are recursively defined from $n, p \in \mathbb{N}$, and the amount of resources needed to build an element of the family is of a polynomial order in n and p, as shown below:
   - Size of the alphabet: $\frac{15n^2p^2}{2}+3n^2p+3n^2+np^2+\frac{35np}{2}+5n+6p+6 \in \Theta(n^2p^2)$.
   - Initial number of membranes: $3 \in \Theta(1)$.
   - Initial number of objects in membranes: $3np + n + 3 \in \Theta(np)$.
   - Number of rules: $\frac{15n^2p^2}{2} + 7n^2p + np^2 + \frac{33np}{2} + 4n + 6p + 4 \in \Theta(n^2p^2)$.
   - Maximal number of objects involved in any rule: $3 \in \Theta(1)$.

(b) The family $\mathbf{\Pi}$ is polynomially bounded with regard to ($\texttt{SAT}$, cod, s): indeed for each instance φ of the $\texttt{SAT}$ problem, any computation of the system $\mathbf{\Pi}(s(\varphi))$ with input multiset $cod(\varphi)$ takes at most $2n + 4np + 2p + 5$ computation steps.

(e) The family $\mathbf{\Pi}$ is sound with regard to ($\texttt{SAT}$, cod, s): indeed for each instance φ of the $\texttt{SAT}$ problem, if the computation of $\mathbf{\Pi}(s(\varphi)) + cod(\varphi)$ is an accepting computation, then φ is satisfiable.

(f) The family $\mathbf{\Pi}$ is complete with regard to ($\texttt{SAT}$, cod, s): indeed, for each instance φ of the $\texttt{SAT}$ problem such that φ is satisfiable, any computation of $\mathbf{\Pi}(s(\varphi)) + cod(\varphi)$ is an accepting computation.

Therefore, the family $\mathbf{\Pi}$ of P systems previously constructed solves the $\texttt{SAT}$ problem in polynomial time and in a uniform way.

Corollary 1. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{DAM}^0(+e_s,mcmp_{out},-d,+n)}$.

Proof. It suffices to notice that $\texttt{SAT}$ problem is a **NP**-complete problem, $\texttt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0(+e_s,mcmp_{out},-d,+n)}$, and the complexity class $\mathbf{PMC}_{\mathcal{DAM}^0(+e_s,mcmp_{out},-d,+n)}$ is closed under polynomial-time reduction and under complement.

6 Conclusions

From a computational complexity point of view and assuming that $\mathbf{P} \neq \mathbf{NP}$, dissolution rules play a crucial role in classical polarizationless P systems with active membranes where there is no cooperation, no changing labels neither priorities. In that framework, **PSPACE**-complete problems can be solved in polynomial time when dissolution rules and division for elementary and non-elementary membranes are permitted. However, dissolution rules and division rules for non-elementary membranes can be replaced by minimal cooperation (the left-hand side of the rules has at most two objects) and minimal production (the right-hand side of the rules has at most two objects) in object evolution rules in order to obtain the computational efficiency [11].

In this paper, the ingredient of minimal cooperation and minimal production in object evolution rules is replaced by minimal cooperation and minimal production in send-out communication rules but we have need to use division for non-elementary membranes. The new systems considered are able to efficiently solve computational hard problems even by considering *simple object evolution rules*, that is, these kind of rules only produce one object. An analogous result can be obtained if minimal cooperation and minimal production are considered only for send-in rules, instead of send-out rules ([12]).

The case where only elementary division is allowed, while keeping the restriction that minimal cooperation and minimal production are used in communication rules of the same direction (only *out* or only *in*) remains as future work, as well as the case where division rules are replaced by separation rules.

What about the class $\mathcal{SAM}^0(+e_s, mcmp_{out}, -d, +n)$? That is, what happens if we revisit the framework studied in this paper but replacing division rules by separation rules? We can adapt the reasoning used in the proof of $\mathbf{P} = \mathbf{PMC}_{\mathcal{SAM}^0_{bmc}(-d,-n)}$ (see [10]), and we can prove that by using families of recognizer membrane systems belonging to this class, only problems in class **P** can be solved in polynomial time.

Acknowledgements

This work was partially supported by Grant numbers 61472328 and 61320106005 of the National Natural Science Foundation of China.

References

1. A. Alhazov, L. Pan. Polarizationless P systems with active membranes. *Grammars*, **7** (2004), 141-159.
2. A. Alhazov, L. Pan, Gh. Păun. Trading polarizations for labels in P systems with active membranes. *Acta Informaticae*, **41**, 2-3 (2004), 111-144.

3. T.H. Cormen, C.E. Leiserson, R.L. Rivest. *An Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1994.
4. M.R. Garey, D.S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
5. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, F.J. Romero–Campero. On the power of dissolution in P systems with active membranes. In R. Freund, Gh. Păun, Gr. Rozenberg, A. Salomaa (eds.). *Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, Lecture Notes in Computer Science*, **3850** (2006), 224–240.
6. Gh. Păun. P systems with active membranes: Attacking **NP**–complete problems, *Journal of Automata, Languages and Combinatorics*, **6** (2001), 75–90. A preliminary version in *Centre for Discrete Mathematics and Theoretical Computer Science Research Reports Series*, CDMTCS-102, May 1999.
7. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
8. P. Sosík, A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, **73** (2007), 137152.
9. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Polarizationless P systems with active membranes: Computational complexity aspects. *Journal of Automata, Languages and Combinatorics*, **21**, 1-2 (2016), 107123
10. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núñez, M.J. Pérez-Jiménez. Minimal cooperation in polarizationless P systems with active membranes. In C. Graciani, Gh. Păun, D. Orellana-Martn, A. Riscos-Nez, L. Valencia-Cabrera (eds.) *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 1-5 February, 2016, Sevilla, Spain, Fénix Editora, pp. 327-356.
11. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Reaching efficiency through collaboration in membrane systems: dissolution, polarization and cooperation. *Theoretical Computer Science*, in press, 2017.
12. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Restricted polarizationless P systems with active membranes: minimal cooperation only inwards. In this volume, 2017 (manuscript).
13. C. Zandron, C. Ferretti, G. Mauri. Solving **NP**-complete problems using P systems. In I. Antoniou, C.S. Calude, M.J. Dinneen (eds.) *Unconventional Models of Computation, UMC'2K*, Springer, London, 2000, pp. 153-164.

Author Index